

Il File System

Il File System

Il file system e' quella parte del Sistema Operativo che fornisce i meccanismi necessari per l'accesso e l'archiviazione delle informazioni in memoria secondaria.

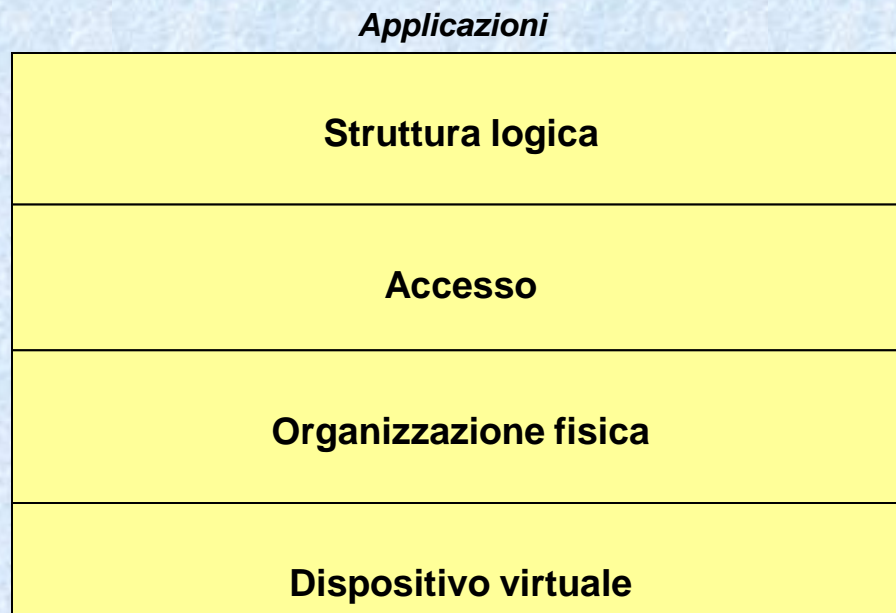
Realizza i concetti astratti:

- ✓ **di file (archivio)**: unità logica di memorizzazione
- ✓ **di directory (cartella)**: insieme di file (e cartelle)
- ✓ **di partizione**: dispositivo virtuale, corrispondente a una porzione di un dispositivo

Le caratteristiche di file, directory e partizione sono del tutto indipendenti dalla natura e dal tipo di dispositivo utilizzato.

6.1 Organizzazione del File System

La struttura di un file system può, in generale, essere rappresentata da un insieme di componenti organizzate in vari livelli:



Hardware: memoria secondaria

Struttura Logica: Il File

Il file è un insieme di informazioni, rappresentate mediante insieme di **record logici** (bit, byte, linee, record, etc.)

Ogni file è caratterizzato da un insieme di **attributi**
ad esempio:

- **tipo**: stabilisce l'appartenenza a una classe (eseguibili, batch, testo, etc)
- **dimensione**: numero di byte contenuti nel file
- **indirizzo**: puntatore/i al record logico corrente (lettura/scrittura)
- **data e ora** (di creazione e/o di modifica)

in S.O. multiutenti anche:

- **utente proprietario**
- **diritti di accesso** al file per il proprietario e gli altri utenti

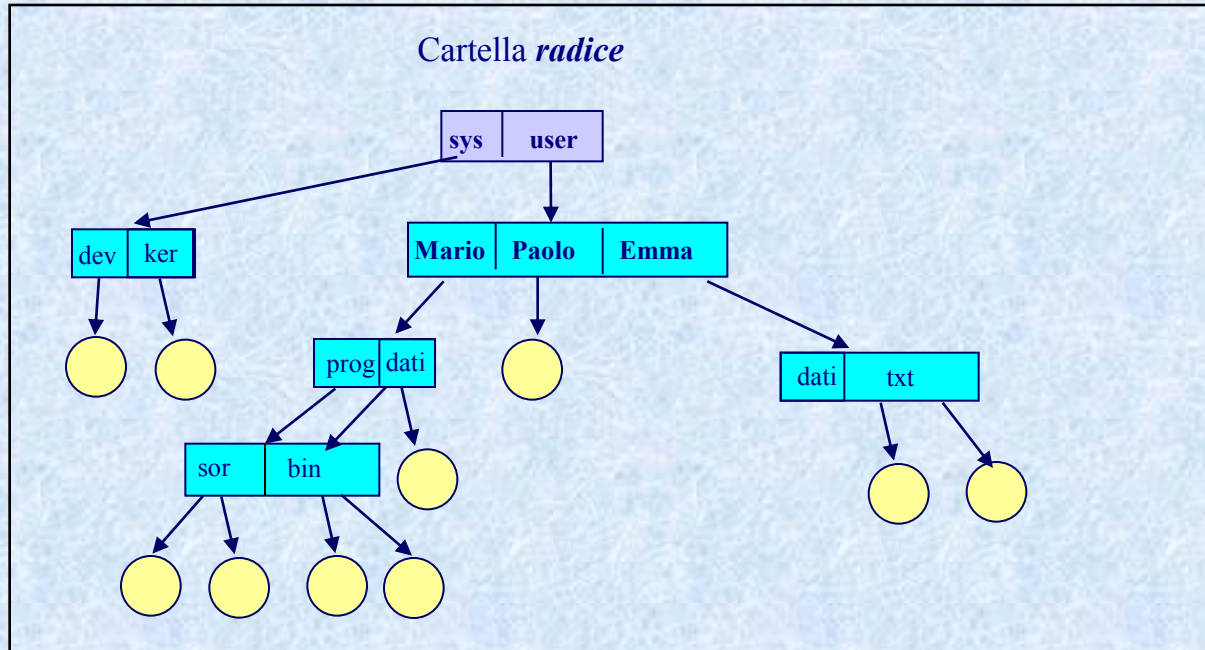
Struttura Logica: La cartella (directory)

E' un'astrazione che raggruppa logicamente più file :

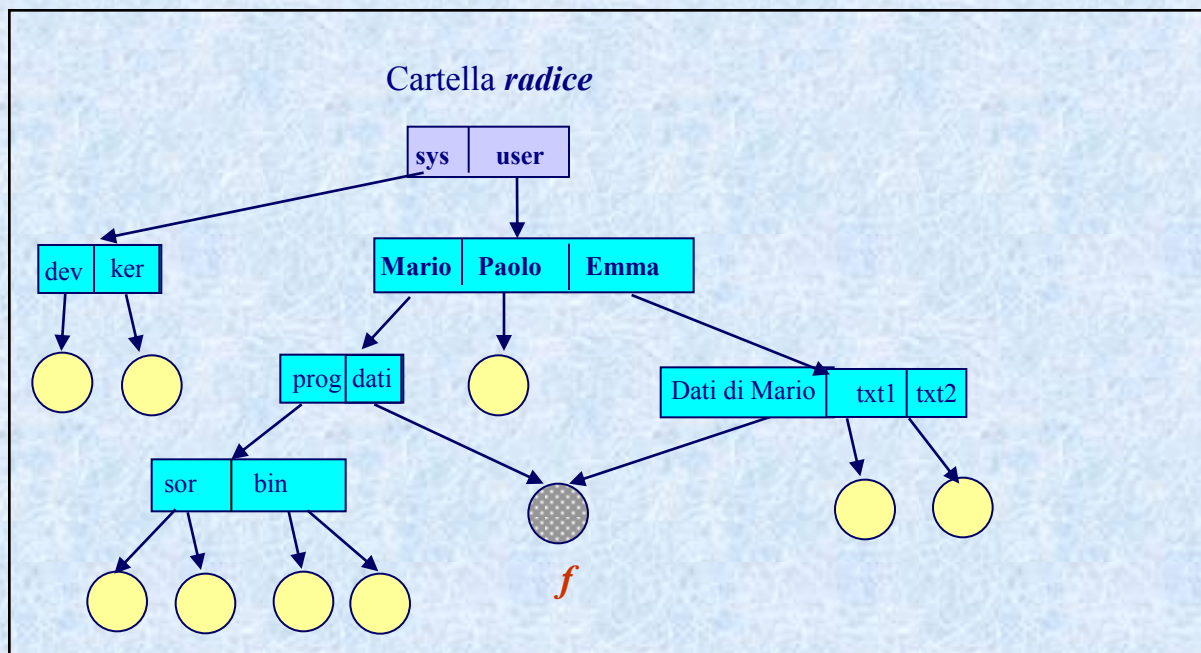
- una cartella può contenere più file
- può contenere a sua volta altre cartelle

➔ La struttura logica del file system è un'aggregazione di file e cartelle, la cui struttura è definita dalle cartelle stesse

File System: struttura logica ad albero



File System: struttura logica a grafo aciclico



--> Collegamento di file

Gestione del File System (operazioni sulle cartelle)

Operazioni fondamentali per la gestione del file system:

- **Creazione e cancellazione di cartelle**
modificano la struttura logica del file system
- **Listing:** ispezione del contenuto di uno (o più) cartelle
- **Attraversamento della cartella:** *navigazione* attraverso la struttura logica del file system
- **Creazione/cancellazione di file**

Accesso al file system

Strutture dati: rappresentazione del file

- ogni file è rappresentato concretamente dal *descrittore di file*, che memorizza gli attributi

Unix: i-node

- i descrittori di file devono essere memorizzati in modo persistente, mediante apposite strutture in memoria secondaria

Unix: i-list

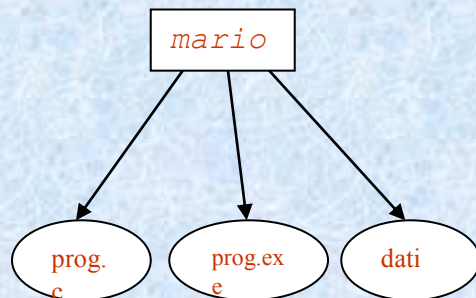
6.3.1 Strutture dati

Cartelle

- Ogni file (eventualmente una cartella) appartiene a una cartella;
- Ogni cartella collega i nomi dei file in essa contenuti ai rispettivi attributi

Possibile realizzazione:

- La cartella è una struttura dati di tipo tabellare che associa ai nomi dei file la lista dei rispettivi attributi, incorporando i descrittori di file :

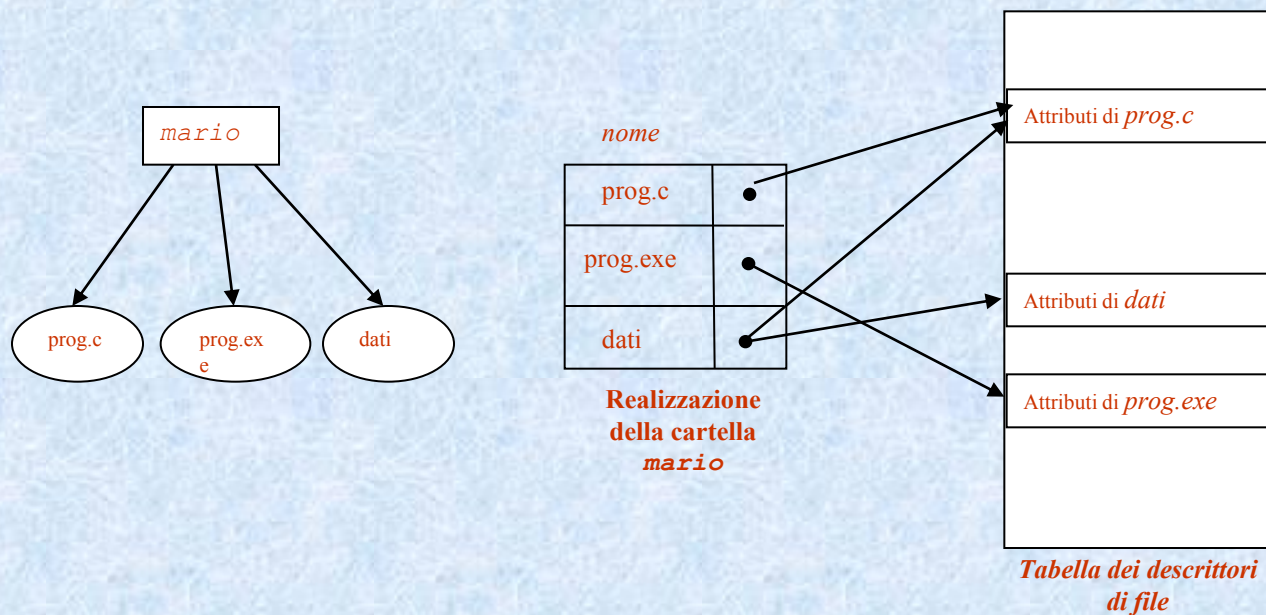


nome	descrittore
prog.c	Attributi: tipo, indirizzi, ecc.
prog.exe	Attributi: tipo, indirizzi, ecc.
dati	Attributi: tipo, indirizzi, ecc.

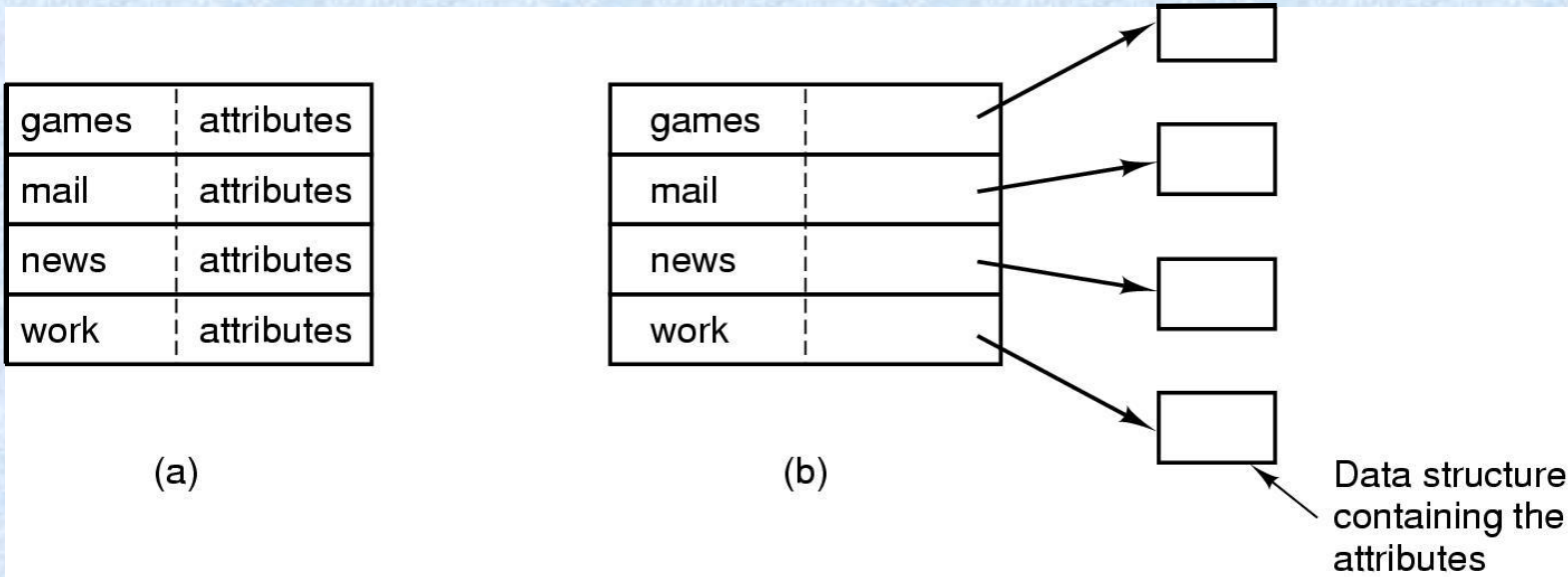
Realizzazione della cartella *mario*

In alternativa (*Unix*):

- la cartella è una tabella che contiene i riferimenti ai descrittori dei file (memorizzati in una struttura separata)



Alternative per l'implementazione delle cartelle



(a) Una semplice directory

- elementi di ampiezza fissa
- contiene sia attributi, sia indirizzi su disco

(b) Una directory che contiene i puntatori alle strutture dati che contengono gli attributi (descrittori di file)

Accesso a file

Compito del S.O. è consentire l'accesso *on-line* ai file.

Operazioni di accesso:

- Lettura di record logici dal file.
- Scrittura aggiunta di record logici al file.

Per eseguire un'operazione necessario reperire gli attributi del file:

- gli indirizzi dei record logici a cui accedere
- Diritti di accesso
-

➔ *costo elevato se gli attributi vengono ricercati nella copia permanente del descrittore di file*

Accesso a file

Per migliorare l'efficienza:

- il S.O. mantiene in memoria una struttura che registra i file attualmente in uso (*file aperti*):
 - per ogni file aperto: copia del descrittore di file
==> posizione su disco, altri attributi
- *Memory mapping* dei file aperti:
 - i file aperti (o porzioni di essi) vengono temporaneamente mappati nello spazio logico del processo
-> accesso più veloce.

Operazioni necessarie:

- **apertura**: introduzione di un nuovo elemento nella tabella dei file aperti
- eventuale **memory mapping** del file.
- **chiusura**: salvataggio del descrittore e dei buffer in memoria secondaria e eliminazione dell'elemento corrispondente dalla tabella dei file aperti.

Metodi di accesso

L'accesso a file può avvenire secondo varie modalità:

- accesso *sequenziale*
- accesso *diretto*
- accesso a *indice*

Il metodo di accesso è indipendente:

- dal tipo di dispositivo **utilizzato**
- dalla tecnica di allocazione **dei blocchi in memoria secondaria.**

Accesso sequenziale

Il file è una sequenza $[R_1, R_2, \dots, R_N]$ di record logici:

- per accedere ad un particolare record logico R_i , è necessario accedere prima agli $(i-1)$ record che lo precedono nella sequenza:



- Operazioni di accesso:
 - ✓ *readnext*: lettura del prossimo record logico della sequenza
 - ✓ *writenext*: scrittura del prossimo record logico
- ogni operazione di accesso (lettura/scrittura) posiziona il puntatore sull'elemento successivo a quello letto/scritto.

Accesso diretto

Il file è un insieme non ordinato $\{R_1, R_2, \dots, R_N\}$ di record logici:

- si può accedere direttamente ad un particolare record logico.
- Operazioni di accesso:
 - ✓ *read(f, i, &V)*: lettura del record logico i del file f ; il valore letto viene assegnato alla variabile V ;
 - ✓ *write(f, i, &V)*: scrittura del valore della variabile V nel record logico i del file f .

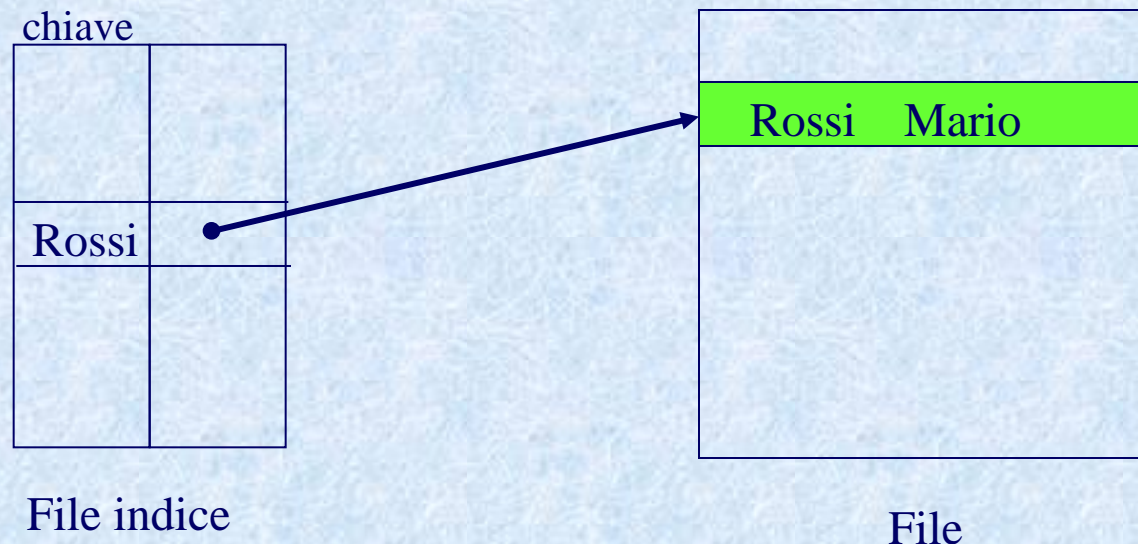
Accesso a indice

Ad ogni file viene associata una struttura dati (un file) contenente l'indice delle informazioni contenute nel file, ciascuna identificata da una *chiave*

Operazioni di accesso:

- ✓ ***read(f, key, &V)***: lettura dal file *f* e trasferimento nella buffer *V* del record logico con chiave uguale a *key*
- ✓ ***write(f, key, V)***: scrittura del contenuto del buffer *V* nel record logico con chiave *key*.

Per accedere a un record logico((ad esempio con il comando ***read(f, "Rossi", &V)***) si esegue una ricerca *per chiave* nell'indice



Protezione

Le politiche di protezione delle risorse (file e direttori) possono essere specificate mediante una **matrice di protezione**:

<i>DOMINIO</i>	<i>File 1</i>	<i>File 2</i>	<i>File 3</i>	<i>File 4</i>	<i>Dir 1</i>	<i>Dir 2</i>	<i>Stamp</i>
Utente Mario	r, w, -	r, -, -				r, w, -	- w, -
Utente Paola				r, w, x	r, -, -		
Utente Elena		r, w, -	r, w, x	r, -, -		r, w, -	- w, -

- Ogni riga corrisponde a un **dominio di protezione**
- Ogni colonna rappresenta una **risorsa** del sistema

Rappresentazione delle politiche di protezione

Due modalità:

- Liste di controllo degli accessi (ACL)
- Liste di capability (C-list)

ACL

La *Access Control List (ACL)* associata alla risorsa *R* elenca i diritti di accesso sulla risorsa *R* attribuiti a ogni dominio

- *rappresenta una colonna della matrice di protezione*

<i>DOMINIO</i>	<i>File 1</i>	<i>File 2</i>	<i>File 3</i>	<i>File 4</i>	<i>Dir 1</i>	<i>Dir 2</i>	<i>Stamp</i>
Utente Mario	r, w, -	r, -, -				r, w, -	- w, -
Utente Paola				r, w, x	r, -, -		
Utente Elena		r, w, -	r, w, x	r, -, -		r, w, -	- w, -

- Ad esempio, la lista di controllo degli accessi associata a *File4* è

File4 --> Paola: (r, w, x) --> Elena: (r, -, -)

C-list

La *Capability List (C-list)* del dominio *D* associa a *D* la lista dei diritti di accesso sulle diverse risorse del sistema attribuiti al dominio *D*.

- *rappresenta una riga della matrice di protezione:*

<i>DOMINIO</i>	<i>File 1</i>	<i>File 2</i>	<i>File 3</i>	<i>File 4</i>	<i>Dir 1</i>	<i>Dir 2</i>	<i>Stamp</i>
Utente Mario	r, w, -	r, -, -				r, w, -	- w, -
Utente Paola				r, w, x	r, -, -		
Utente Elena		r, w, -	r, w, x	r, -, -		r, w, -	- w, -

Ad esempio, la C-lista associata all'utente Mario è la seguente:

Utente Mario --> File1:(r, w, -) --> File2:(r, -, -) --> Dir2:(r, w, -) --> Stamp:(-, w, -)

Organizzazione fisica del File System

Ogni dispositivo di memoria secondaria è organizzato in *blocchi* (o *record fisici*):

Blocco: unità di trasferimento nelle operazioni di I/O da/verso il dispositivo; la sua dimensione è costante.

Ma: i processi vedono ogni file come un insieme di *record logici*:

Record logico: unità logica di trasferimento specificata nelle operazioni accesso al file.

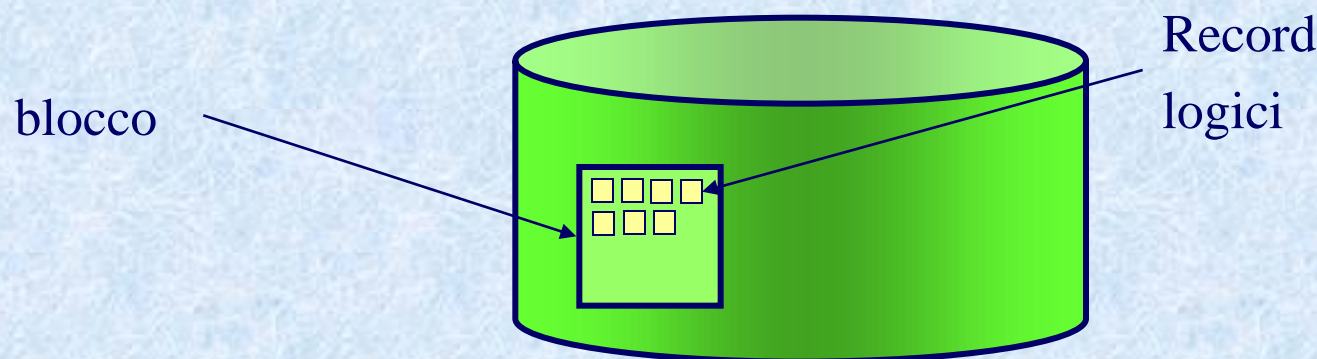
6.4.1 Blocchi & record logici

Uno dei compiti del file system è stabilire una corrispondenza tra record logici e blocchi fisici.

Di solito:

Dimensione(blocco) >> Dimensione(record logico)

➔ ogni blocco fisico può contenere più record logici.



Metodi di Allocazione dei file

Ogni *blocco fisico* contiene un insieme di *record logici* contigui.

==> *record logici* raggruppati in *blocchi logici*, ciascuno di dimensione uguale a quella del *blocco fisico*

==> *allocazione* dei blocchi fisici: assegna un blocco fisico a ciascun blocco logico

Principali metodi di allocazione dei blocchi fisici

- **allocazione** contigua
- **allocazione** a lista
- **allocazione** a indice

Allocazione contigua

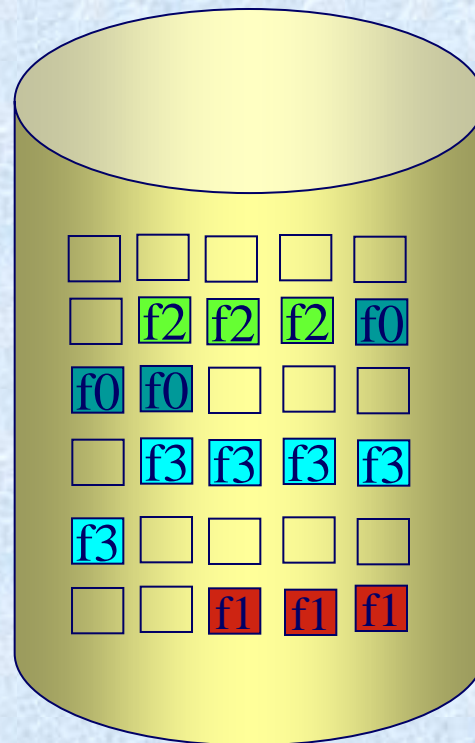
Ogni file occupa un insieme di blocchi fisici fisicamente contigui.

Vantaggi:

- ✓ Efficienza dell'accesso sequenziale
- ✓ Facilità della ricerca di un blocco e dell'accesso diretto

Svantaggi:

- ✓ Frammentazione esterna:
man mano che si riempie il disco, le regioni contigue libere sono sempre più piccole e possono risultare inutilizzabili per l'allocazione dei nuovi file
- ✓ Complessità dell'allocazione
- ✓ Ostacoli all'espansione dei file
- ✓ *si può ovviare con la compattazione (deframmentazione) del disco*



Allocazione a lista concatenata (1) inizio

I blocchi fisici sui quali viene mappato ogni file sono organizzati in una lista concatenata, con puntatori interni ai blocchi medesimi

f2		232
----	--	-----

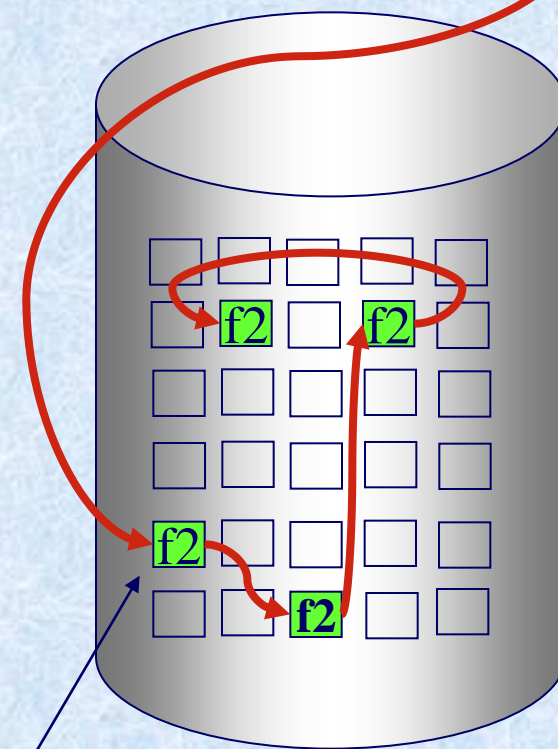
Cartella o Descrittore di File

Vantaggi:

- ✓ non c'è frammentazione esterna
- ✓ Allocazione facile

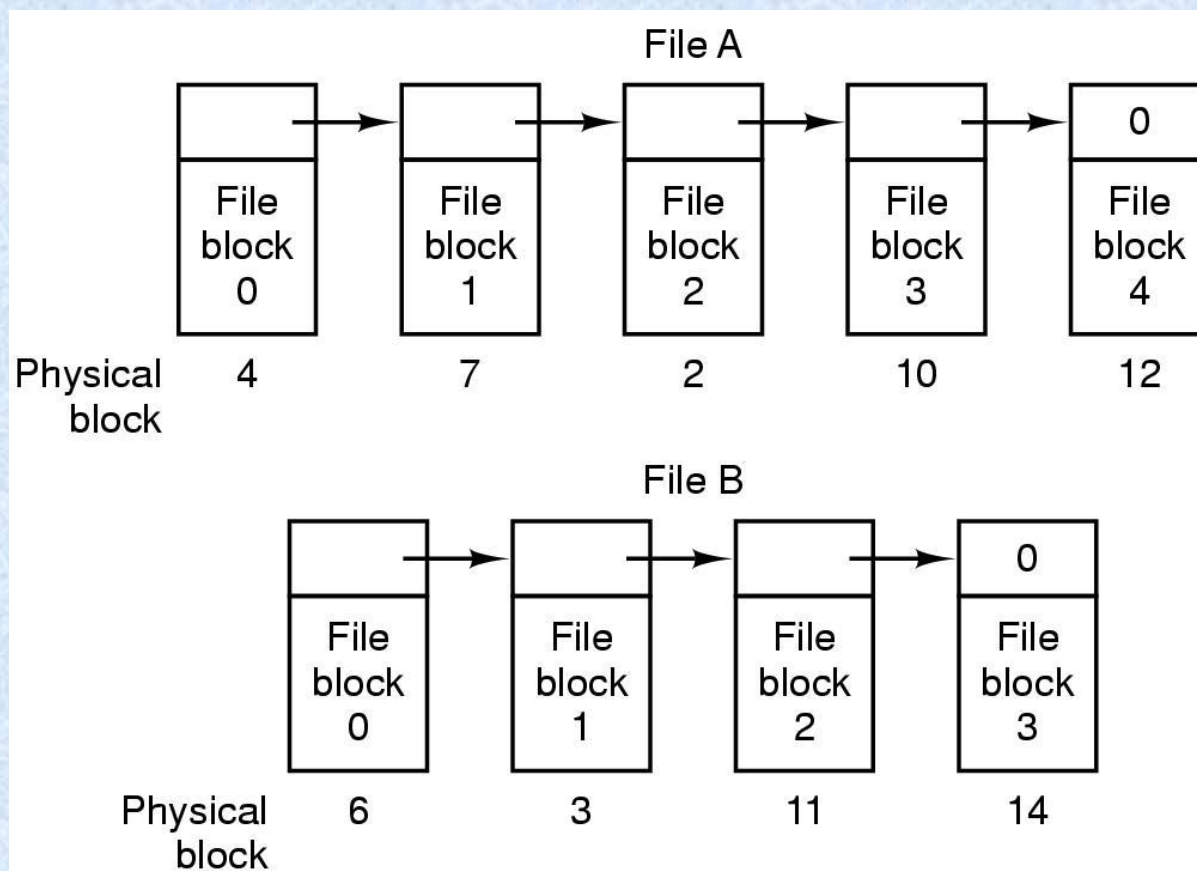
Svantaggi:

- ✓ costo della ricerca di un blocco
- ✓ accesso diretto inefficiente
- ✓ possibilità di perdere blocchi se un link viene danneggiato
- ✓ maggior occupazione (spazio occupato dai puntatori)



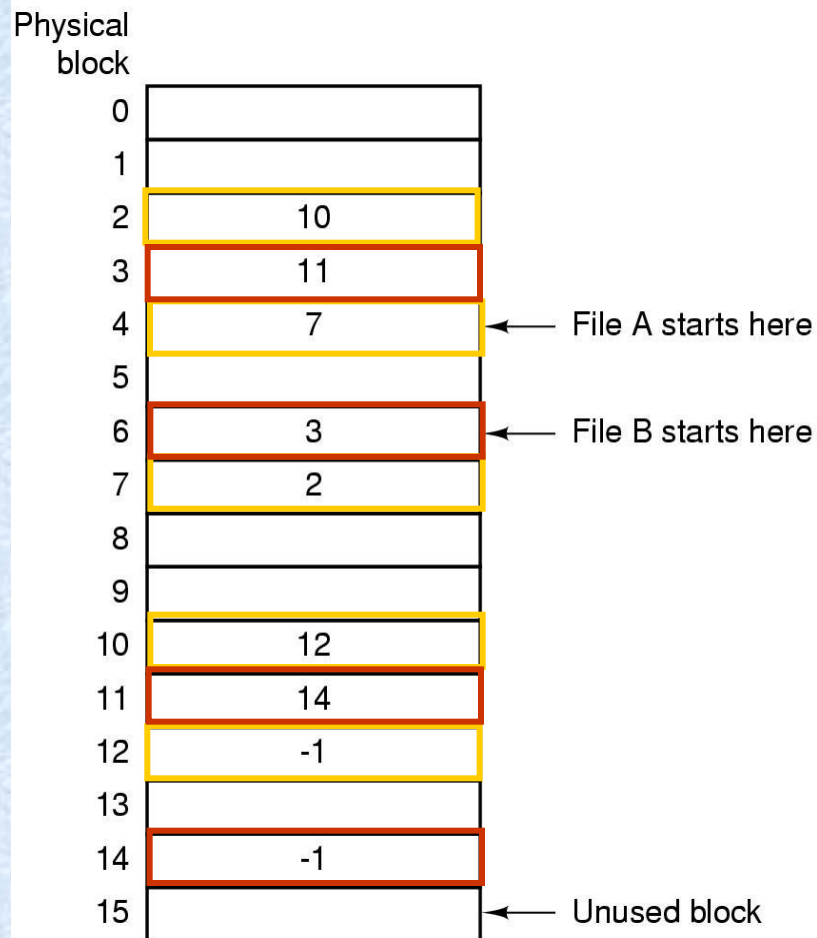
blocco 232

Allocazione a lista concatenata (2)



Allocazione con *File Allocation Table (FAT)*

E' una evoluzione dell'allocazione con lista concatenata



- I puntatori che collegano i blocchi fisici sono memorizzati come elementi di una tabella (*File Allocation Table*), separata dai blocchi

- A ogni blocco fisico corrisponde un elemento della *FAT*

- La *FAT* risiede permanentemente su disco, ma una copia di lavoro è caricata in memoria

- Ricerca di un blocco: si deve percorrere una lista di elementi della *FAT*, ma gli accessi avvengono in memoria invece che su disco

==> accesso diretto efficiente

La *FAT*:

- Unica tabella che descrive tutti i file (aperti o no)

- grande dimensione, superabile con il caricamento dinamico (paginazione a domanda)

- Gli elementi che descrivono un file sono generalmente sparsi
 - nell'accesso diretto (= ricerca di un record logico) sono possibili molti errori di pagina

Limitazioni dei file systems FAT

Posto:

- L lunghezza (in bit) degli elementi della FAT
- B la lunghezza (in byte) dei blocchi del disco,
- il numero di blocchi indirizzabili è 2^L (= capacità del disco, o partizione)
--> la massima estensione del file system è 2^L blocchi, ovvero a $B * 2^L$ byte.
- se ogni elemento occupa N byte (solitamente L è multipla del byte), la FAT occupa complessivamente $N * 2^L$ byte

Esempio: con $N=2$ (--> *FAT 16*): $B = 2^{10}$:

- La massima estensione del file system è 2^{16} blocchi, ovvero 2^{26} byte (= 64 Mbyte)
- la FAT occupa complessivamente $2 * 2^{16}$ byte = 128 Kbyte
- con una memoria paginata e pagine di 1Kbyte, la FAT occupa 128 pagine
- dato che gli elementi che descrivono un file possono essere distribuiti su molte pagine diverse, possono verificarsi frequenti errori di pagina quando si percorre un file.

--> Per realizzare file systems più estesi: gli elementi della FAT indirizzano blocchi logici, multipli del blocco fisico del disco.

Limitazioni dei file systems FAT

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Massima dimensione del File System per diverse ampiezze dei blocchi

Allocazione a indice

A ogni file sono associati uno o più blocchi
(*blocchi indice o blocchi indiretti*), ciascuno di
quale contiene indirizzi di blocchi fisici su cui è
allocato il file.

Vantaggi:

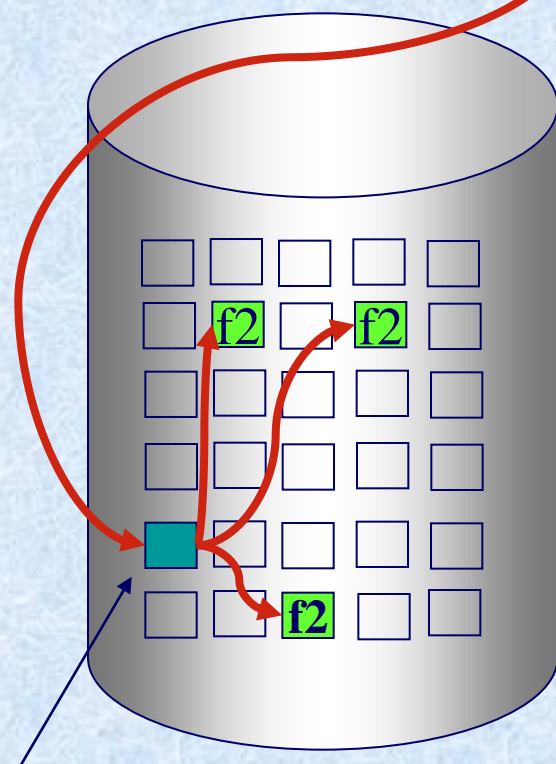
- ✓ assenza di frammentazione esterna
 - ✓ facilità di allocazione
 - ✓ facilità di accesso diretto
 - ✓ strutture dati separate per file diversi
- possono essere caricate in memoria con
limitata occupazione*

Svantaggi:

- ✓ Frammentazione interna nei blocchi indice

indice		
f2		124

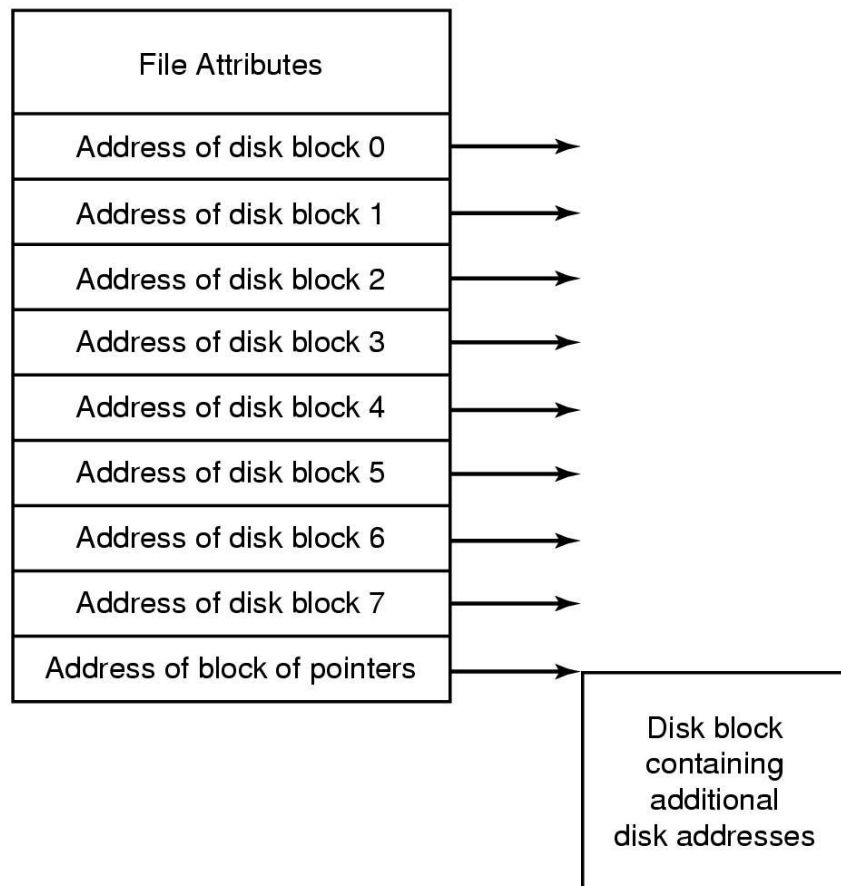
Cartella o Descrittore di File



Blocco indice n. 124

Allocazione a indice (2)

Schema semplificato di Unix



- i-node utilizzato (parzialmente) come blocco indice
- altri blocchi indice individuati dagli indirizzi indiretti

Metodi di Allocazione

Riassumendo, gli aspetti caratterizzanti sono:

- grado di utilizzo della memoria
- tempo di accesso medio al blocco
- realizzazione dei metodi di accesso

- Alcuni sistemi operativi che adottano più di un metodo di allocazione; spesso:
 - file piccoli → allocazione contigua
 - file grandi → allocazione a indice

Il File System di UNIX

Omogeneità

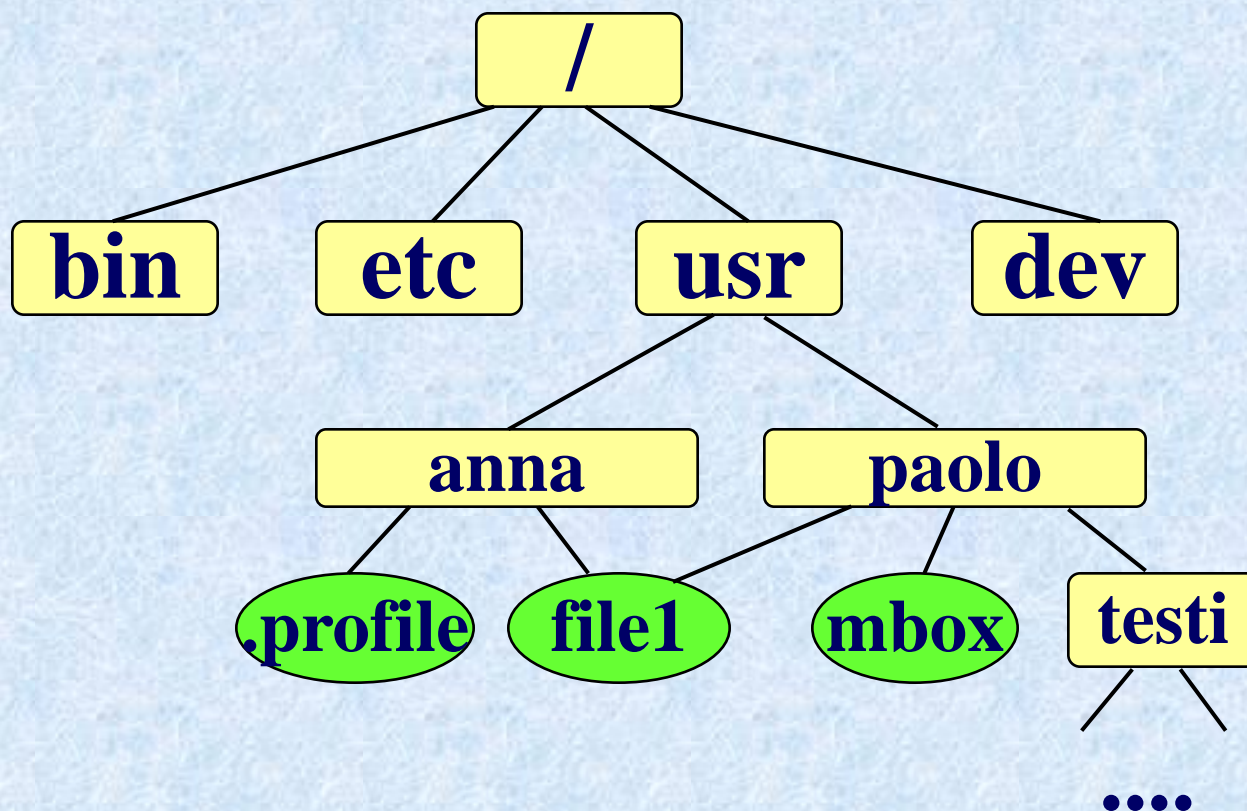
- Astrazione dei dispositivi e dei pipe come file

Tre categorie di file:

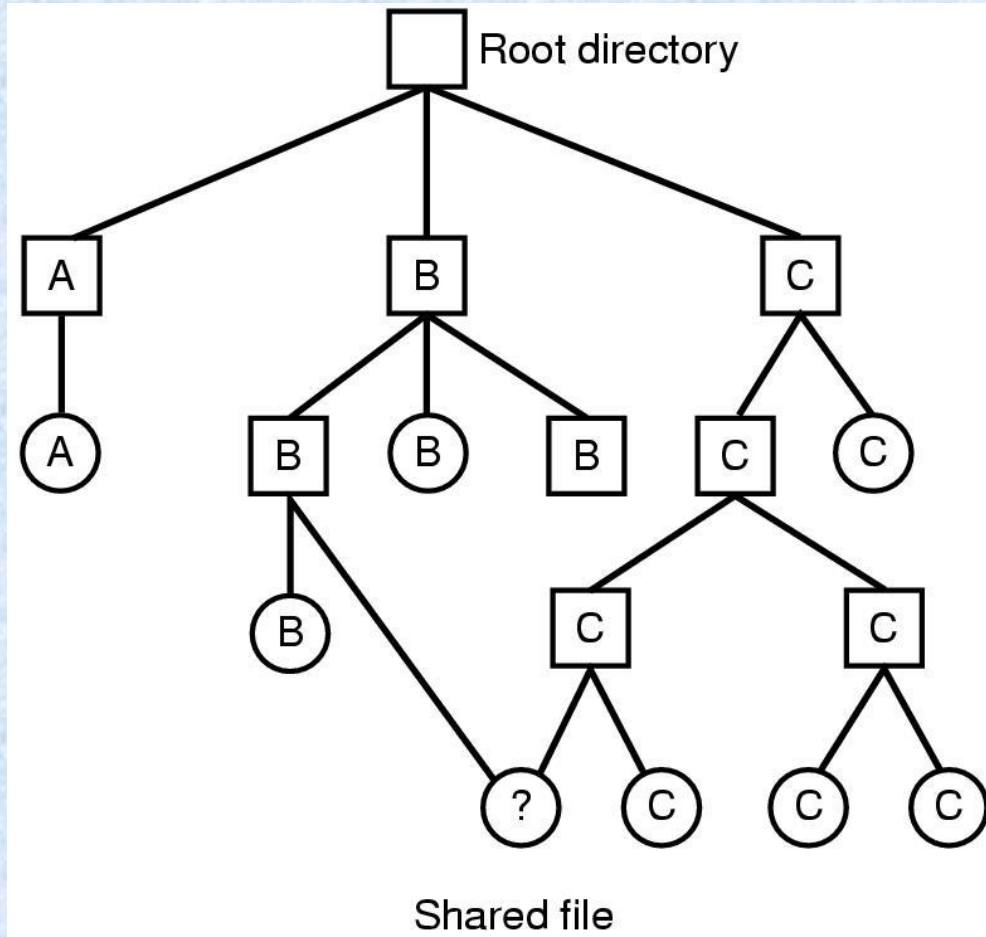
- file ordinari
- cartelle
- file speciali (dispositivi, pipe)

Il File System di Unix

Organizzazione logica: grafo aciclico



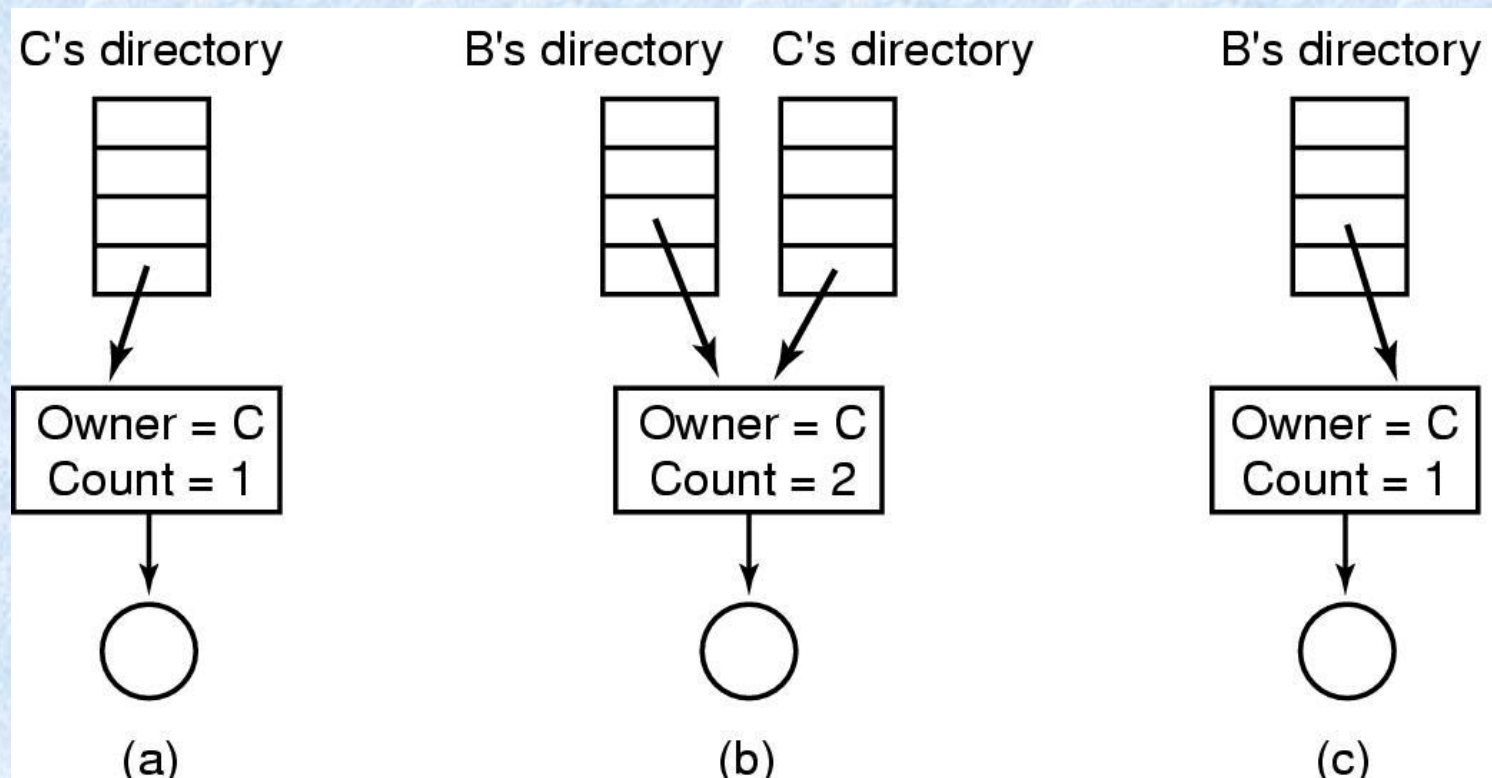
Collegamento (linking) di file (1)



1) collegamento *rigido*

2) collegamento *simbolico*

Collegamento rigido (linking) di file (2)

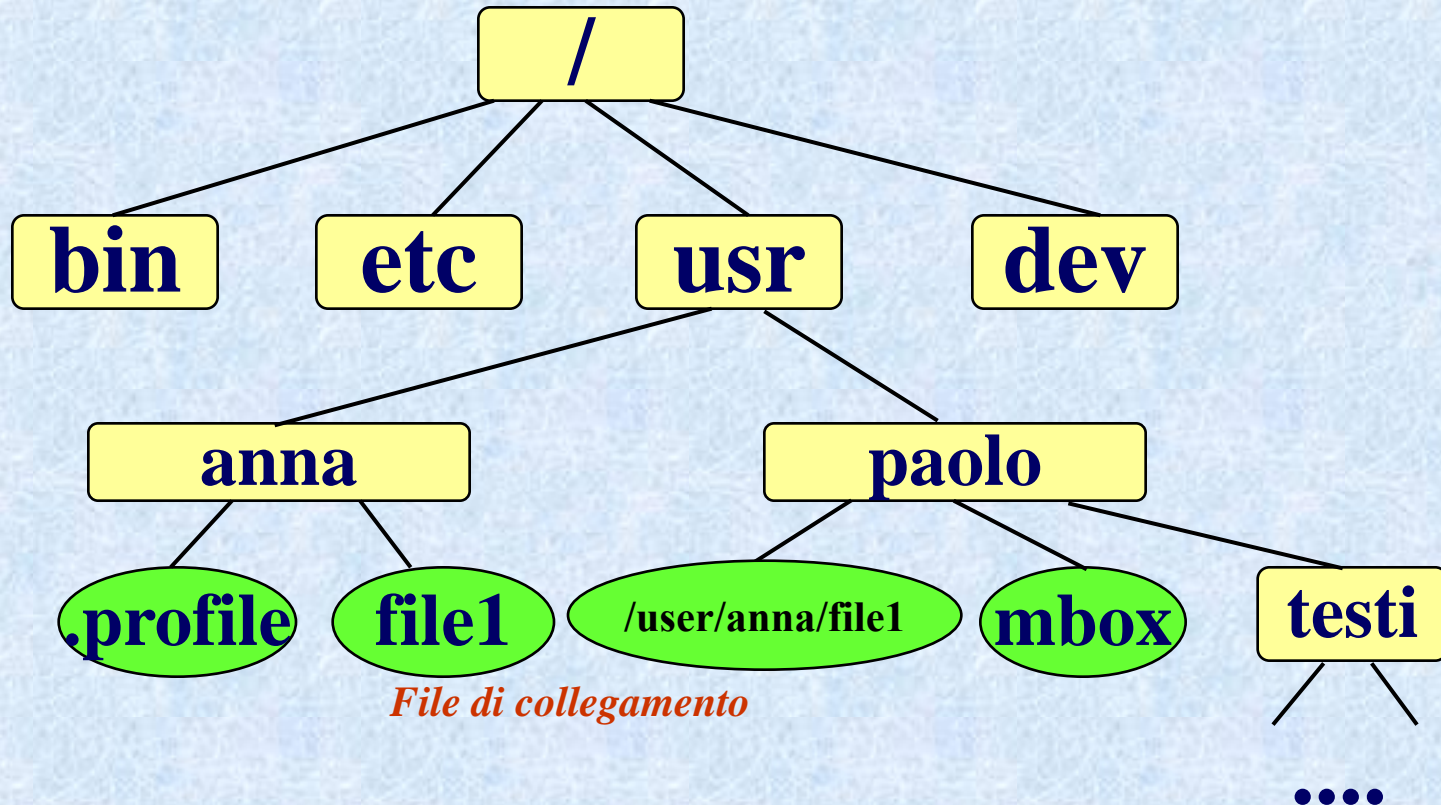


(a) situazione precedente al collegamento ($C = \text{owner}$)

(b) dopo il collegamento da parte di B

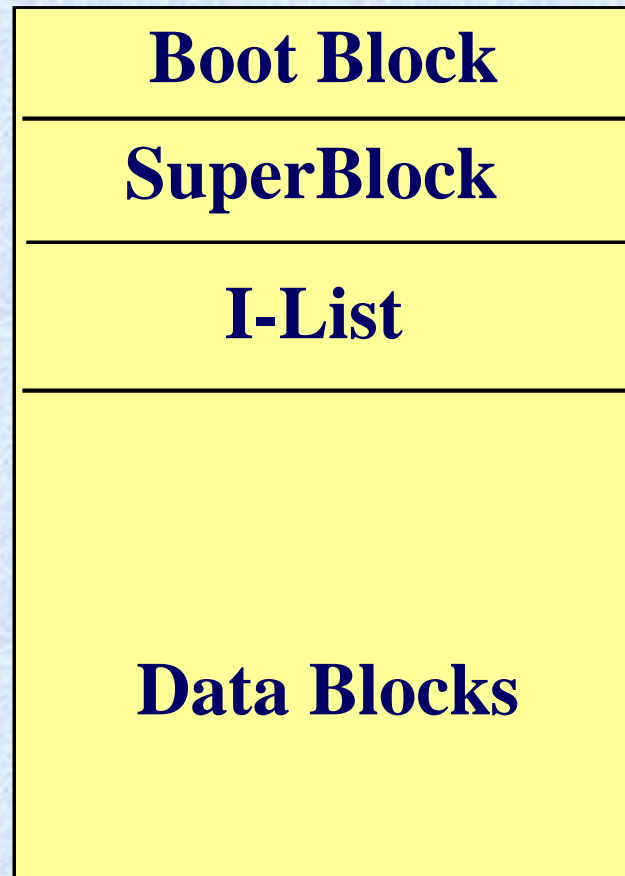
(c) dopo che lo *owner* C ha rimosso il file (*unlink*)

Collegamento *simbolico* di file

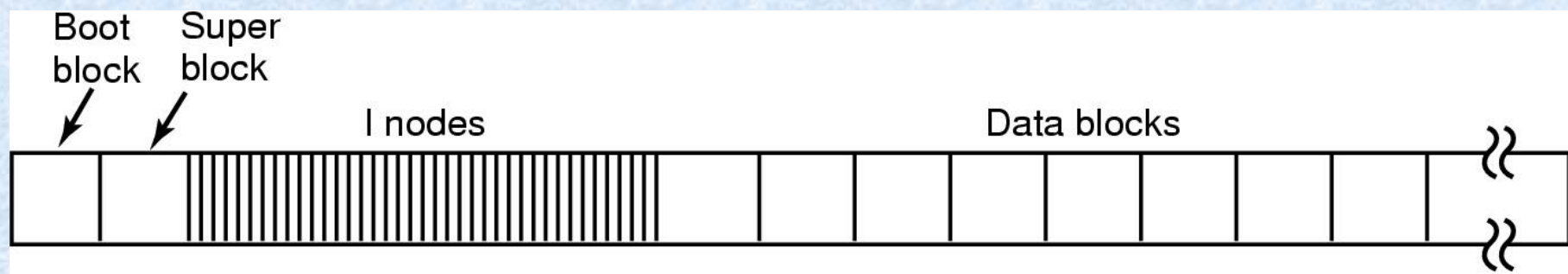


Il File System di Unix: organizzazione fisica

- **Informazione residente su disco**



Organizzazione fisica del disco nei sistemi UNIX



i-node

È il **descrittore** del file.

Tra gli attributi contenuti nell'*i-node*:

- **tipo di file:**
 - ✓ ordinario
 - ✓ directory
 - ✓ file speciale
- proprietario, gruppo (*user-id, group-id*)
- Lunghezza del file
- Data e tempo di creazione e modifica
- 12 bit di **protezione**
- numero di **links**
- **13 - 15 indirizzi di blocchi** (a seconda della realizzazione)

Struttura di un i-node

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

La protezione in UNIX

Si applica in modo uniforme ai file ordinari, cartelle e dispositivi

Basato su una ACL semplificata

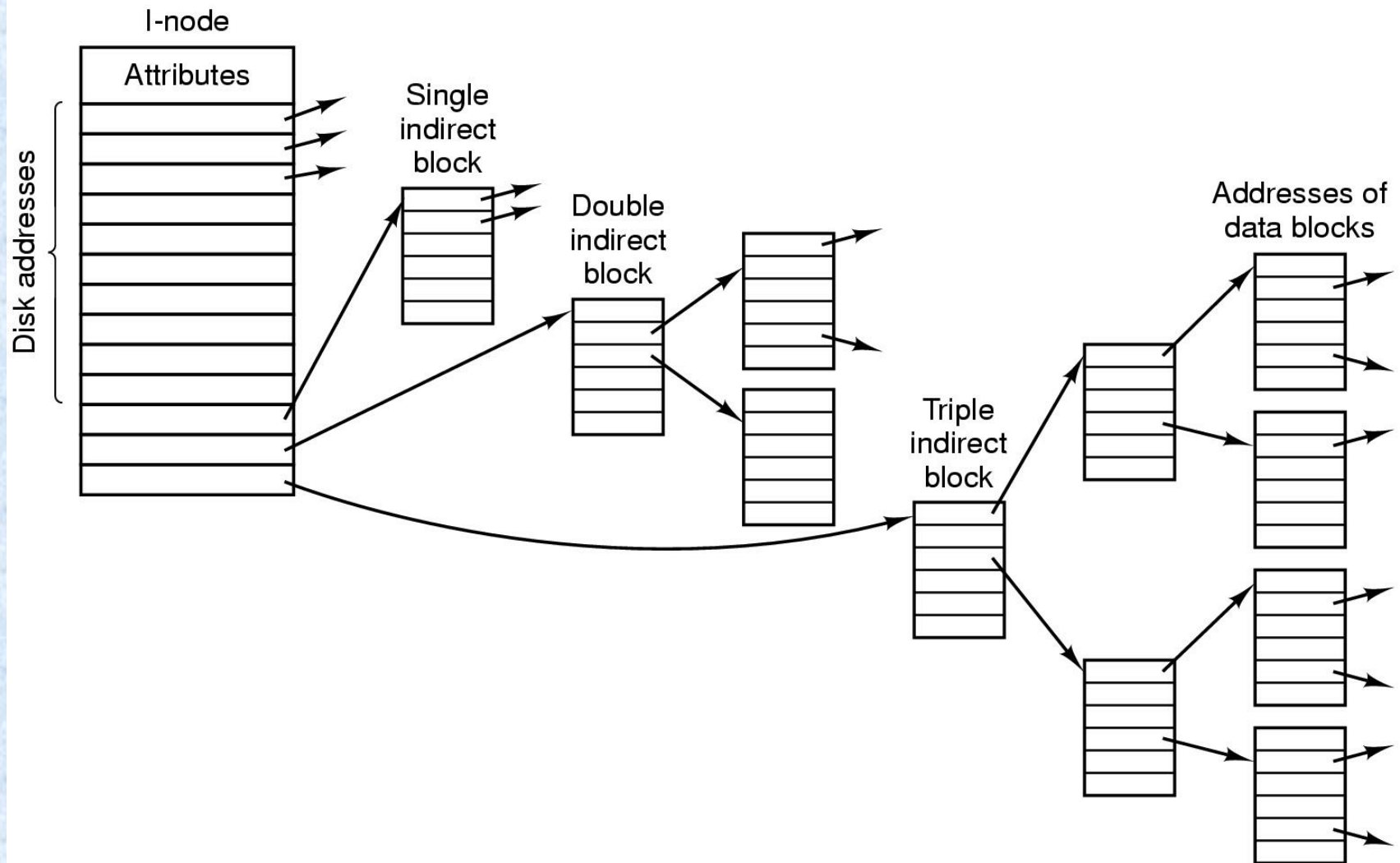
DOMINIO	TERNA DI PROTEZIONE
Proprietario	r, w, x
Gruppo	r, w, x
Altri	r, w, x
	SetUID, SetGID, STicky

- *r: diritto di lettura*
- *w: diritto di scrittura*
- *x: diritto di esecuzione (nel caso delle cartelle: attraversamento)*

SetUID, SetGID, STicky: si applicano ai file eseguibili

- *SetUID: il processo che esegue assume i diritti del proprietario*
- *SetGID: il processo che esegue assume i diritti del gruppo*
- *STicky: Save Text Image*

Indirizzamento dei blocchi tramite i-node e blocchi indiretti



Indirizzamento dei blocchi tramite i-node e blocchi indiretti

	Puntatori diretti										Puntatori indiretti		
IndicePuntatore	0	1	2	3	4	5	6	7	8	9	10	11	12
puntatore	0	0	0							0			

DEFINIZIONI:

BloccoLogico: indice di un blocco logico nel file

BloccoFisico: indice di un blocco nel disco

IPOTESI

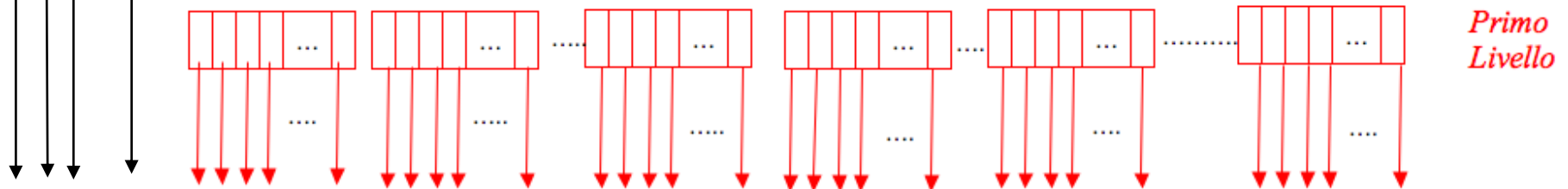
Nello i-node: 10 indirizzi diretti, 3 indirizzi indiretti

Lunghezza di ogni blocco (logico e fisico): 1 Kbyte

Lunghezza di ogni puntatore: 4 byte

=> numero di puntatori in ogni blocco indiretto: $2^{10}/4 = 2^8$

Blocchi Indice, ovvero Blocchi Indiretti di Primo Livello



10 blocchi fisici indirizzabili con puntatori diretti

if *BloccoLogico* < 10

BloccoFisico = &puntatore[*BloccoLogico*]

if *BloccoLogico* ≥ 10 {

ind1 = (*BloccoLogico* - 10) **div** 2^8 ; *offset1* = (*BloccoLogico* - 10) **mod** 2^8

/* *ind1* è il puntatore a un blocco indiretto di primo livello */

BloccoFisico = &(&*ind1*[*offset1*])

}

Come indirizzare i Blocchi Indiretti di Primo Livello?

Il primo blocco indiretto di primo livello (--> *ind1* = 0) è indirizzato da puntatore[10] (puntatore indiretto semplice)

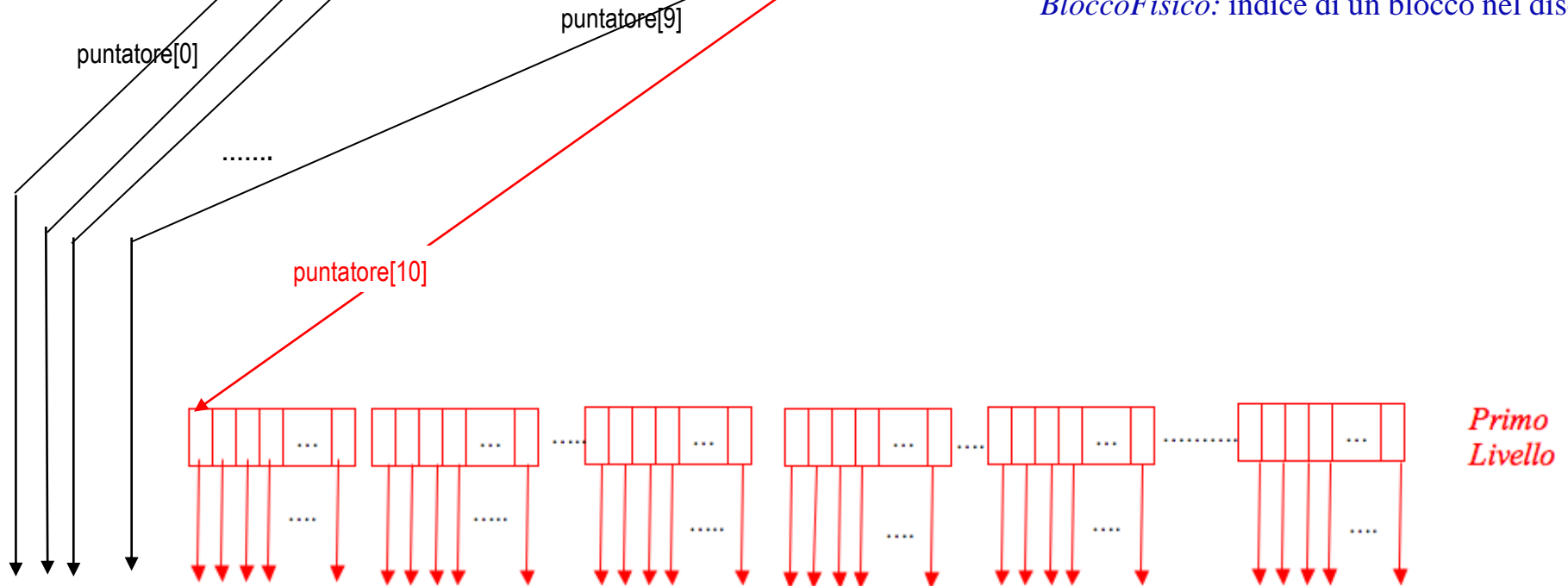
Indirizzamento dei blocchi tramite i-node e blocchi indiretti

	Puntatori diretti										Puntatori indiretti		
IndicePuntatore	0	1	2	3	4	5	6	7	8	9	10	11	12
puntatore	0	0	0							0	●		

DEFINIZIONI:

BloccoLogico: indice di un blocco logico nel file

BloccoFisico: indice di un blocco nel disco



10 blocchi fisici indirizzabili con puntatori diretti

2⁸ blocchi fisici indirizzabili con puntatore indiretto semplice;

1 accesso al disco per leggere il blocco indiretto di primo livello

```

if BloccoLogico ≥ 10 {
    ind1 = (BloccoLogico - 10) div 28; offset1 = (BloccoLogico - 10) mod 28
    /* ind1 è il puntatore a un blocco indiretto di primo livello */
    if ind = 0 BloccoFisico = &(&puntatore[10])[offset1]
}
    
```

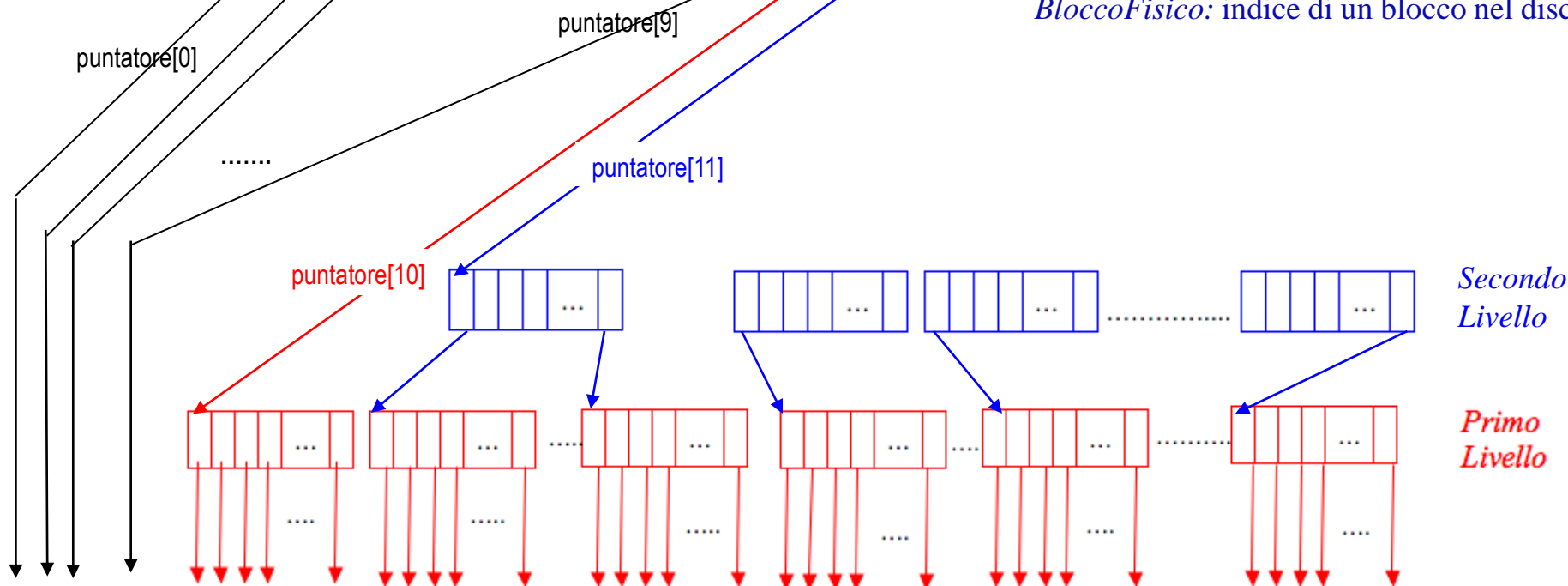
Indirizzamento dei blocchi tramite i-node e blocchi indiretti

	Puntatori diretti										Puntatori indiretti		
IndicePuntatore	0	1	2	3	4	5	6	7	8	9	10	11	12
puntatore	○	○	○							○	●	●	

DEFINIZIONI:

BloccoLogico: indice di un blocco logico nel file

BloccoFisico: indice di un blocco nel disco



10 blocchi fisici indirizzabili con puntatori diretti

2^8 blocchi fisici indirizzabili con puntatore indiretto semplice;

2^{16} blocchi fisici indirizzabili con puntatore indiretto doppio;

2 accessi al disco, per leggere un blocco indiretto di secondo livello e un blocco indiretto di primo livello

```

if BloccoLogico ≥ 10 {
    ind1 = (BloccoLogico - 10) div 28; offset1 = (BloccoLogico - 10) mod 28;
    /* ind1 è il puntatore a un blocco indiretto di primo livello */
    if ind1 = 0 BloccoFisico = &(&puntatore[10])[offset1]
    else {
        ind2 = (ind1 - 1) div 28; offset2 = (ind1 - 1) mod 28
        /* ind2 è il puntatore a un blocco indiretto di secondo livello */
        if ind2 = 0 BloccoFisico = &(&(&puntatore[11])[offset2])[offset1]
    }
}
    
```

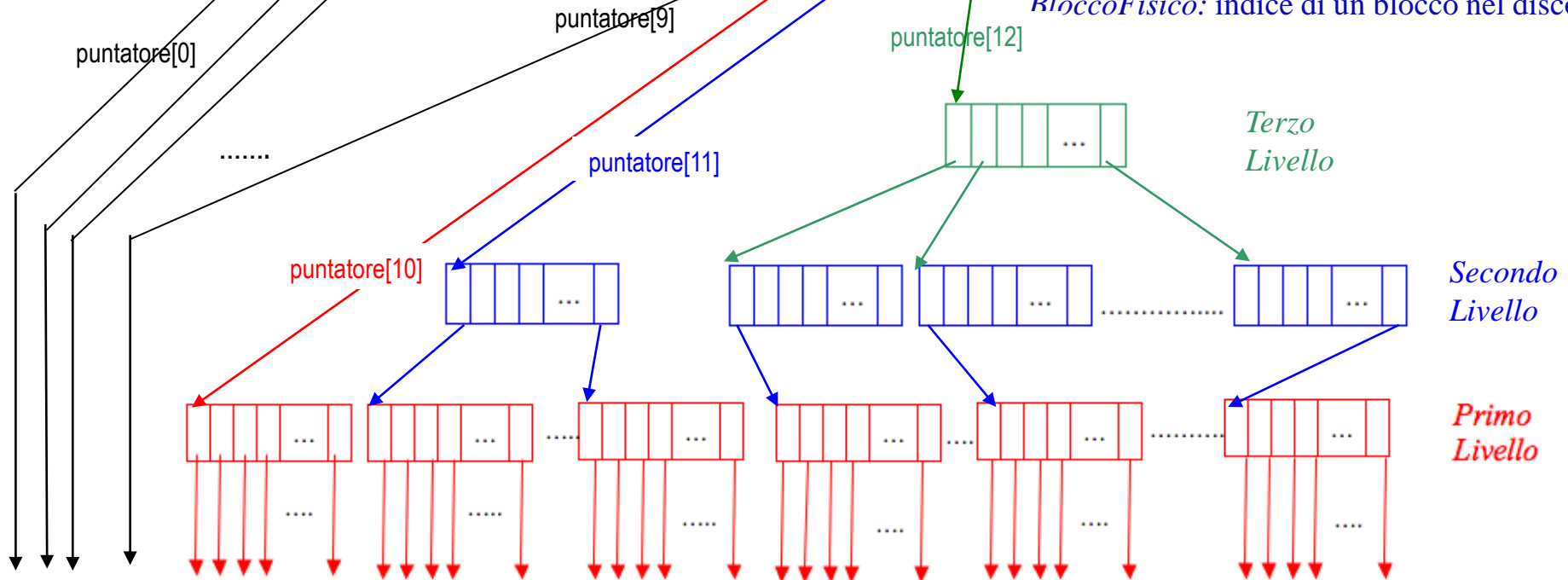

Indirizzamento dei blocchi tramite i-node e blocchi indiretti

	Puntatori diretti										Puntatori indiretti		
IndicePuntatore	0	1	2	3	4	5	6	7	8	9	10	11	12
puntatore	○	○	○							○	●	●	●

DEFINIZIONI:

BloccoLogico: indice di un blocco logico nel file

BloccoFisico: indice di un blocco nel disco



10 blocchi fisici indirizzabili con puntatori diretti

2^8 blocchi fisici indirizzabili con puntatore indiretto semplice;

2^{16} blocchi fisici indirizzabili con puntatore indiretto doppio;

2^{24} blocchi fisici indirizzabili con puntatore indiretto triplo;

3 accessi al disco, per leggere 3 blocchi indiretti, di terzo, di secondo e di primo livello

.....

$ind2 = (ind1 - 1) \div 2^8$; $offset2 = (ind1 - 1) \bmod 2^8$

if $ind2 = 0$ *BloccoFisico* = $\&(\&(\&puntatore[11])[offset2])[offset1]$

else {

$ind3 = 0$; $offset3 = (ind2 - 1) \bmod 2^8$;

/ ind3 è il puntatore all'unico blocco indiretto di terzo livello**

BloccoFisico = $\&(\&(\&(\&puntatore[12])[offset3])[offset2])[offset1]$

Indirizzamento dei blocchi tramite i-node e blocchi indiretti

DEFINIZIONI:

BloccoLogico: indice di un blocco logico nel file; *BloccoFisico*: indice di un blocco nel disco

IPOTESI

Nello i-node: 10 indirizzi diretti, 3 indirizzi indiretti

Lunghezza di ogni blocco (logico e fisico): 1 Kbyte

Lunghezza di ogni puntatore: 4 byte \Rightarrow numero di puntatori in ogni blocco indiretto: $2^{10}/4 = 2^8$

10 blocchi fisici indirizzabili con puntatori diretti; accesso immediato dallo i-node

2⁸ blocchi fisici indirizzabili con puntatore indiretto semplice; 1 accesso al disco per leggere un blocco indiretto di primo livello

2¹⁶ blocchi fisici indirizzabili con puntatore indiretto doppio; 2 accessi al disco, per leggere due blocchi indiretti, di secondo e di primo livello

2²⁴ blocchi fisici indirizzabili con puntatore indiretto triplo; 3 accessi al disco, per leggere 3 blocchi indiretti di terzo, di secondo e di primo livello

```
if BloccoLogico < 10 BloccoFisico = &puntatore[BloccoLogico]
else {
    ind1 = (BloccoLogico - 10) div 28; offset1 = (BloccoLogico - 10) mod 28;
    /* ind1 è il puntatore a un blocco indiretto di primo livello */
    if ind1 = 0 BloccoFisico = &(&puntatore[10])[offset1]
    else {
        ind2 = (ind1 - 1) div 28; offset2 = (ind1 - 1) mod 28;
        /* ind2 è il puntatore a un blocco indiretto di secondo livello */
        if ind2 = 0 BloccoFisico = &(&(&puntatore[11])[offset2])[offset1]
        else {
            ind3 = 0; offset3 = (ind2 - 1) mod 28;
            /* ind3 è il puntatore all'unico blocco indiretto di terzo livello */
            BloccoFisico = &(&(&(&puntatore[12])[offset3])[offset2])[offset1]
        }
    }
}
```

Indirizzamento dei blocchi tramite i-node e blocchi indiretti: esempio

In un file system UNIX i blocchi del disco hanno ampiezza di 1Kbyte e i puntatori ai blocchi sono a 32 bit.

Gli i-node contengono, oltre agli altri attributi, 10 puntatori diretti e 3 puntatori indiretti.

Il primo blocco logico del file e il primo blocco del disco hanno indice 0.

Massima estensione del File System, in base al numero di blocchi indirizzabili nel disco: $2^{32} \text{ blocchi} \rightarrow 2^{32} * 2^{10} = 2^{42} \text{ byte (4 Tbyte)}$

Massima lunghezza di ogni singolo file, in base al numero di blocchi indirizzabili per ogni file:

Considerato che:

- lo i-node indirizza direttamente 10 blocchi
- ogni blocco indiretto contiene $2^{10} \text{ div } 4 = 2^8$ indirizzi
- il blocco indiretto di primo livello puntato dall'indirizzo indiretto semplice indirizza 2^8 blocchi dati
- il blocco indiretto di secondo livello puntato dall'indirizzo indiretto doppio indirizza 2^8 blocchi indiretti di primo livello, ciascuno dei quali indirizza 2^8 blocchi dati \Rightarrow possono essere indirizzati 2^{16} blocchi dati
- il blocco indiretto di terzo livello puntato dall'indirizzo indiretto triplo indirizza 2^8 blocchi indiretti di secondo livello, ciascuno dei quali indirizza 2^8 blocchi indiretti di primo livello, ciascuno dei quali indirizza 2^8 blocchi dati \Rightarrow possono essere indirizzati 2^{24} blocchi dati

la massima lunghezza di ogni singolo file è :

$$10 + 2^8 + 2^{16} + 2^{24} = 10 + 256 + 65.536 + 16.777.216 = 16.843.018 \text{ blocchi}$$

\rightarrow approssimativamente: $16 \text{ Gbyte} \leq \text{massima lunghezza} < 17 \text{ Gbyte}$

Primo BloccoLogico al quale si applica l'indirizzamento diretto: BloccoLogico 0

Ultimo BloccoLogico al quale si applica l'indirizzamento diretto : BloccoLogico 9;

Primo BloccoLogico al quale si applica l'indirizzamento indiretto semplice: BloccoLogico 10

Ultimo BloccoLogico al quale si applica l'indirizzamento indiretto semplice : BloccoLogico $10 + 2^8 - 1 = 265$;

Primo BloccoLogico al quale si applica l'indirizzamento indiretto doppio: BloccoLogico $10 + 2^8 = 266$

Ultimo BloccoLogico al quale si applica l'indirizzamento indiretto doppio : BloccoLogico $10 + 2^8 + 2^{16} - 1 = 65.801$

Primo BloccoLogico al quale si applica l'indirizzamento indiretto triplo: BloccoLogico $10 + 2^8 + 2^{16} = 65.802$

Ultimo BloccoLogico al quale si applica l'indirizzamento indiretto triplo : BloccoLogico $(10 + 2^8 + 2^{16} + 2^{24}) - 1 = 16.843.017$

Indirizzamento dei blocchi tramite i-node e blocchi indiretti: esempio

Considerato il file (aperto) individuato dal file descriptor fd , la cui lunghezza corrente (in byte) è 278.538 e il cui i -node contiene i seguenti puntatori a blocchi:

Puntatore	0	1	2	3	4	5	6	7	8	9	10	11	12
Valore del puntatore	100	101	102	120	121	122	300	301	302	303	500	700	--

dove i blocchi indiretti 500, 700, e 800 hanno i seguenti contenuti parziali:

Blocco 500:

Indice di elemento nel blocco	0	1	2	3	4	5
Valore del puntatore	304	305	306	307	308	309

Blocco 700:

Indice di elemento nel blocco	0	1	2	3	4	5
Valore del puntatore	800	801	802	850	851	852

Blocco 800:

Indice di elemento nel blocco	0	1	2	3	4	5
Valore del puntatore	1200	1201	1202	1203	1204	1205

Ipotesi 1: si deve eseguire l'operazione $read(fd, \&buf, 1)$ quando lo I/O pointer ha valore 12.298

- Il carattere 12.298 è contenuto nel $BloccoLogico = 12.298 \div 2^{10} = 12$.
- Poiché $BloccoLogico \geq 10$, dall'espressione $ind1 = (BloccoLogico - 10) \div 2^8$; $offset1 = (BloccoLogico - 10) \bmod 2^8$ si ha $ind1 = (12 - 10) \div 256 = 0$; $offset1 = (12 - 10) \bmod 256 = 2$;
- Dall'espressione $BloccoFisico = \&(\&puntatore[10])[offset1]$ si ha $puntatore[10] = 500$; $\&500[2] = 306$; $BloccoFisico = 306$

Indirizzamento dei blocchi tramite i-node e blocchi indiretti: esempio

Considerato il file (aperto) individuato dal file descriptor fd , la cui lunghezza corrente (in byte) è 278.538 e il cui i -node contiene i seguenti puntatori a blocchi:

Puntatore	0	1	2	3	4	5	6	7	8	9	10	11	12
Valore del puntatore	100	101	102	120	121	122	300	301	302	303	500	700	--

dove i blocchi indiretti 500, 700, e 800 hanno i seguenti contenuti parziali:

Blocco 500:

Indice di elemento nel blocco	0	1	2	3	4	5
Valore del puntatore	304	305	306	307	308	309

Blocco 700:

Indice di elemento nel blocco	0	1	2	3	4	5
Valore del puntatore	800	801	802	850	851	852

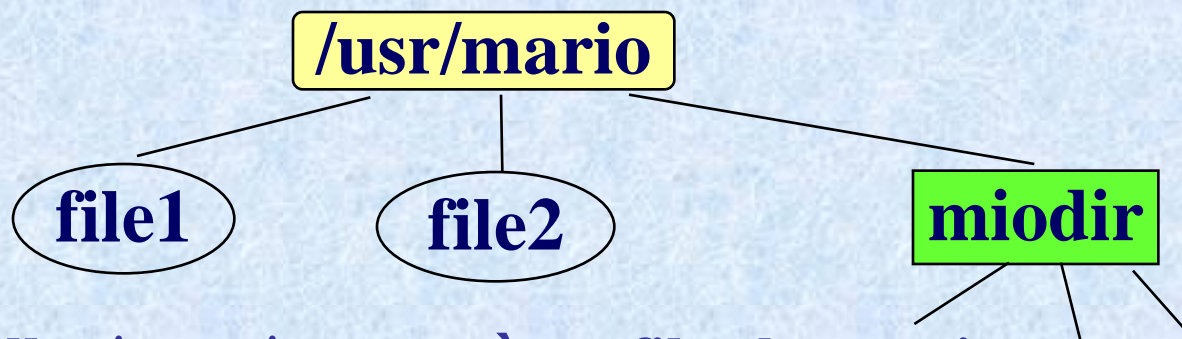
Blocco 800:

Indice di elemento nel blocco	0	1	2	3	4	5
Valore del puntatore	1200	1201	1202	1203	1204	1205

Ipotesi 2: si deve eseguire l'operazione $read(fd, \&buf, 1)$ quando lo I/O pointer ha valore 273.428

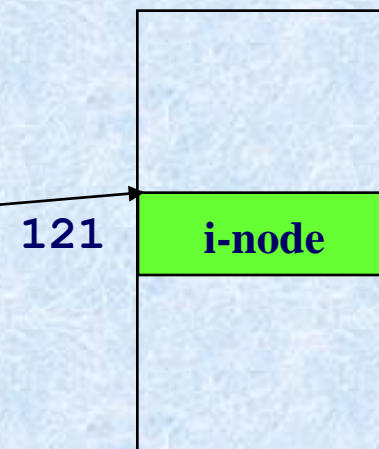
- Il carattere 273.438 è contenuto nel $BloccoLogico = 273.438 \div 2^{10} = 267$.
- Poiché $BloccoLogico \geq 10$, dall'espressione $ind1 = (BloccoLogico - 10) \div 2^8$; $offset1 = (BloccoLogico - 10) \bmod 2^8$ si ha $ind1 = (267 - 10) \div 256 = 1$; $offset1 = (267 - 10) \bmod 256 = 1$;
- Poiché $ind1 > 0$, dall'espressione $ind2 = (ind1 - 1) \div 2^8$; $offset2 = (ind1 - 1) \bmod 2^8$ si ha $ind2 = (1 - 1) \div 256 = 0$; $offset2 = (1 - 1) \bmod 256 = 0$
- Poiché $ind2 = 0$, dall'espressione $BloccoFisico = \&((\&puntatore[11])[offset2]))[offset1]$ si ha $puntatore[11] = 700$; $\&700[0] = 800$; $\&800[1] = 1201$; $BloccoFisico = 1201$

La cartella



La cartella `/usr/mario` è un file che contiene:

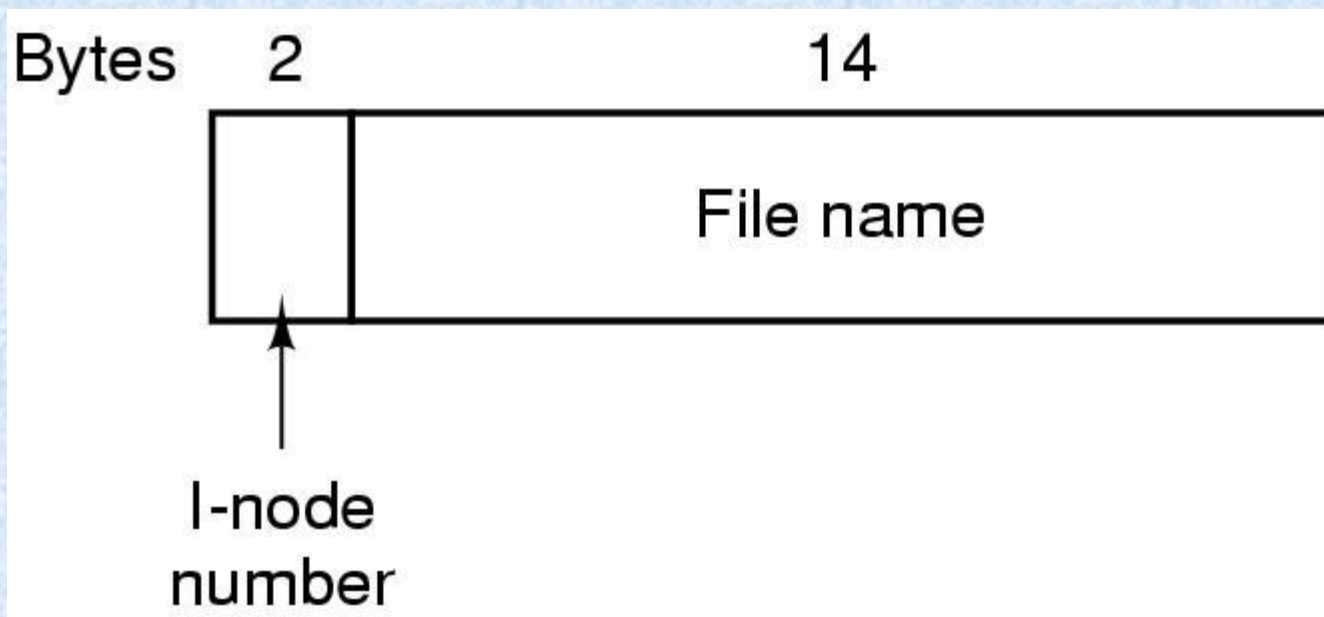
<code>file1</code>	189
<code>file2</code>	133
<code>miodir</code>	121
<code>.</code>	110
<code>..</code>	89



i-list

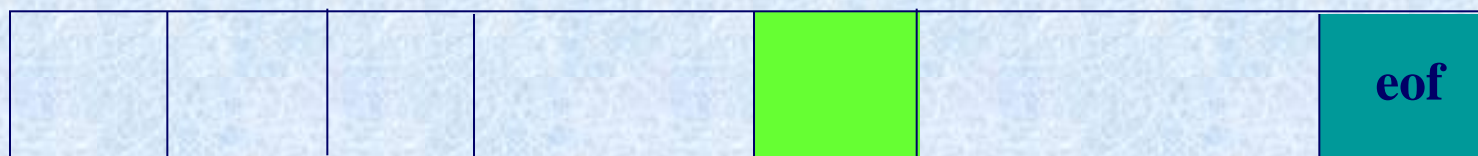
Il File System di UNIX V.7 (1)

Contenuto di un elemento di una directory in UNIX V7



7.6.4 Accesso a File: Concetti Generali

- assenza di strutturazione: *file = sequenza di bytes*
- accesso sequenziale a partire dalla *posizione corrente*
- posizione corrente: ***I/O Pointer***

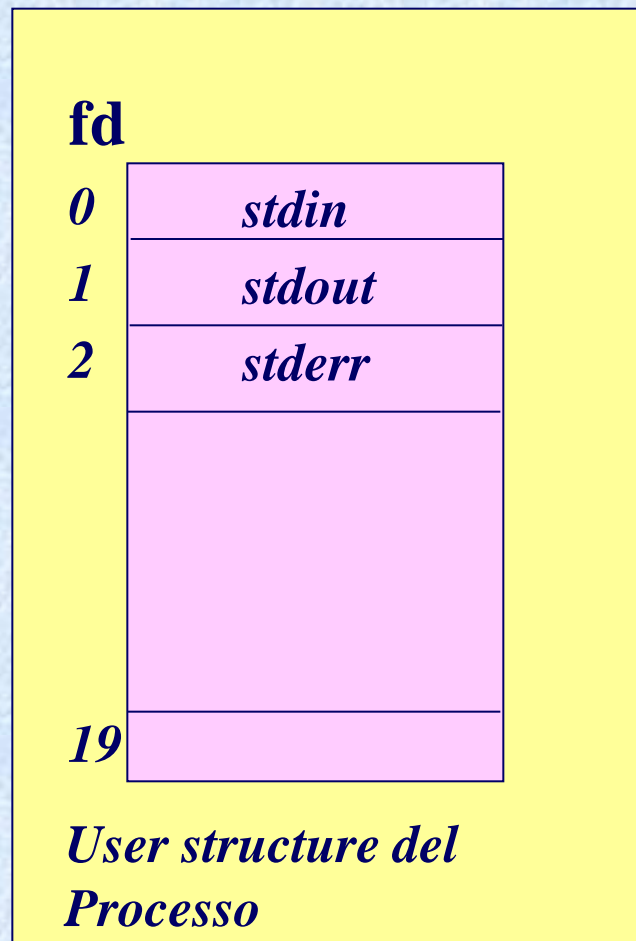


***I/O
pointer***

- **vari modi di accesso** (*lettura, scrittura, lettura/scrittura, etc.*)
- accesso subordinato all'operazione di **apertura**

Strutture dati del Kernel per l'accesso a file (1)

- A ogni processo è associata una *tabella dei file aperti di processo (file descriptor table)*
- ogni elemento della tabella rappresenta un file aperto dal processo, individuato da un indice intero: *file descriptor*
- i file descriptor 0,1,2 individuano rispettivamente *standard input, output, error* (aperti automaticamente)
- la tabella dei file aperti da un processo è contenuta nella sua *user structure*



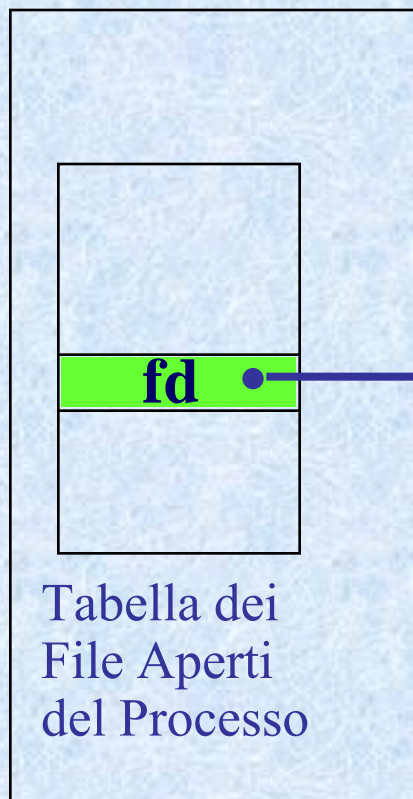
Strutture dati del kernel per l'accesso a file (2)

Oltre alla tabella dei *file aperti di processo*:

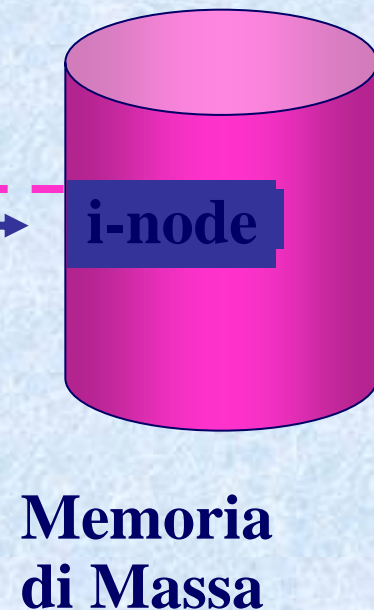
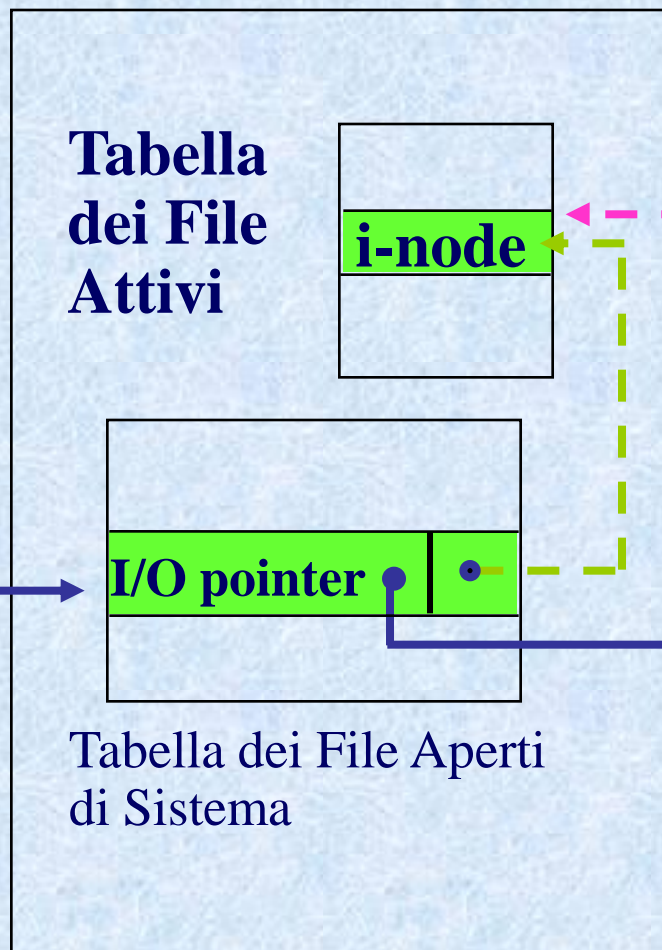
- tabella dei file aperti di sistema (*open file descriptor table*)
- tabella dei file attivi (*i-node table*)

Quali sono le relazioni tra queste strutture?

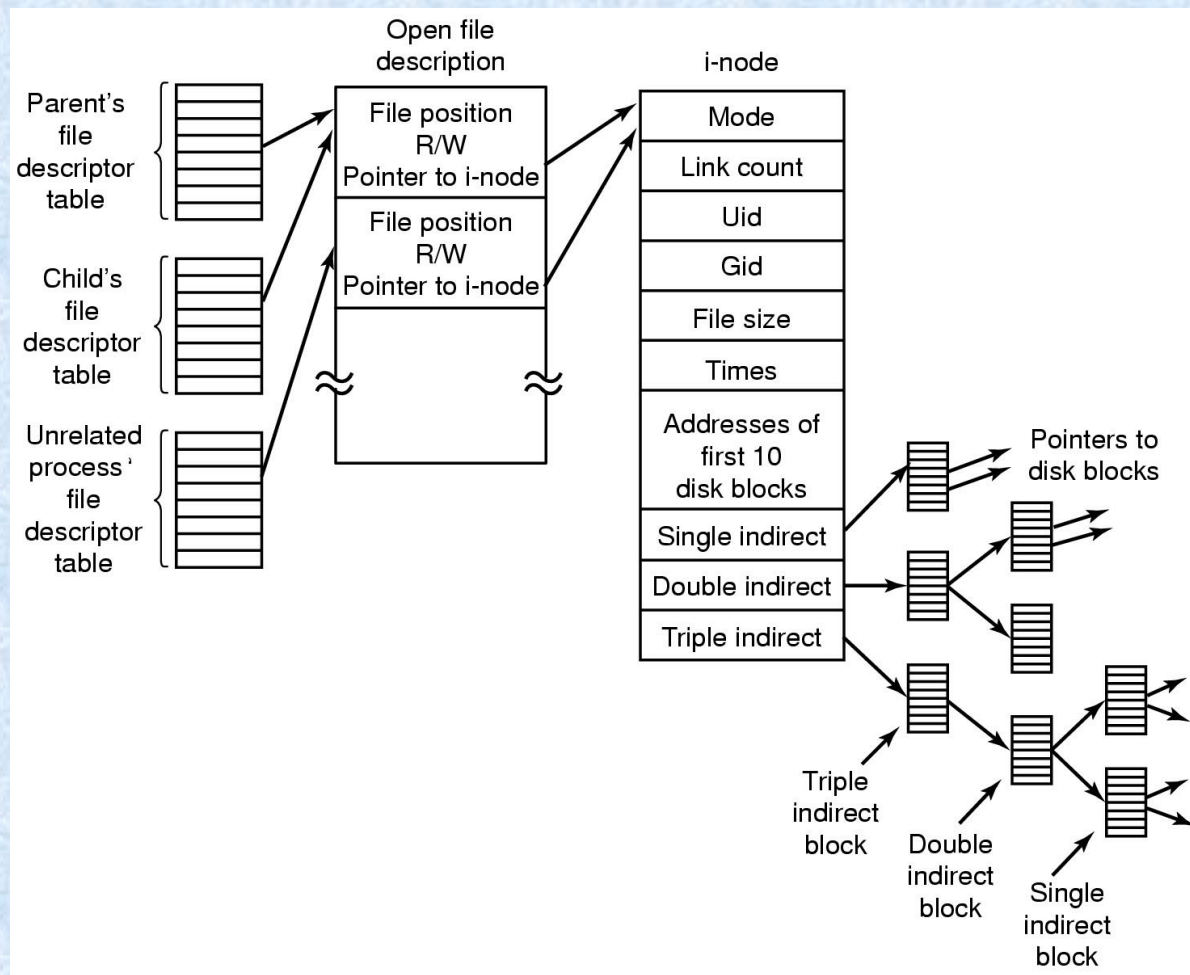
User Area del Processo



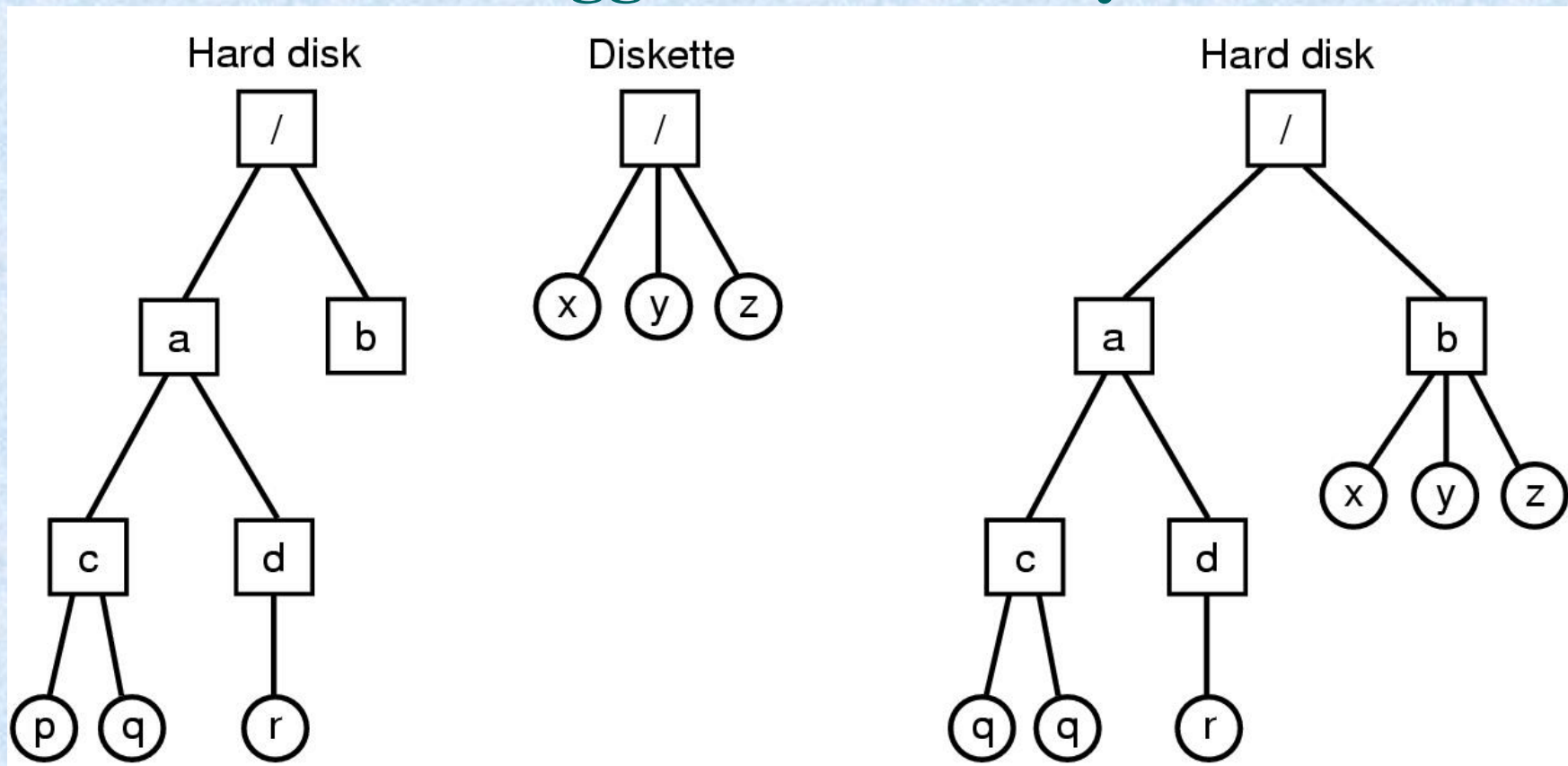
Area Dati del kernel



Relazione fra *file descriptor* e *open file descriptor*



Montaggio di un File System



7.6.5 System Call per l'accesso a file

Unix permette ai processi di accedere a file, mediante un insieme di *system call*, tra le quali:

- ✓ apertura/creazione: `open`, `creat`
- ✓ chiusura: `close`
- ✓ lettura: `read`
- ✓ scrittura: `write`
- ✓ cancellazione: `unlink`
- ✓ linking: `link`
- ✓ accesso diretto: `lseek`

Apertura di File: open

```
int open(char nomefile[],int flag, [int mode]);
```

- **nomefile** è il nome del file (relativo o assoluto)
 - **flag** esprime il modo di accesso (ad es. **O_RDONLY**, per accesso in lettura, **O_WRONLY**, per accesso in scrittura)
 - **mode** è un parametro richiesto soltanto se l'opzione di apertura determina la **creazione** del file(**O_CREAT**): in tal caso, **mode** specifica i bit di protezione
- il valore restituito dalla **open** è il *file descriptor* associato al file, o -1 in caso di errore.

Chiusura di File: `close`

Per chiudere un file aperto:

```
int close(int fd) ;
```

- `fd` è il file descriptor del file da chiudere.
- Restituisce l'esito della operazione (0 in caso di successo, <0 in caso di insuccesso).
- Se la `close` ha successo: i buffer vengono salvati sul disco e vengono eliminati gli elementi associati al file nelle tabelle del kernel.

Lettura e Scrittura di File

Caratteristiche:

- accesso mediante il file descriptor
- ogni operazione di accesso (lettura o scrittura) agisce sequenzialmente sul file, a partire dalla posizione corrente del puntatore (I/O pointer)
- atomicità della singola operazione.
- operazioni sincrone, cioè con attesa del completamento dell'operazione.
- possibilità di alternare operazioni di lettura e scrittura

Lettura di File: system call `read`

```
int read(int fd, char *buf, int n) ;
```

- `fd` è il file descriptor del file
- `buf` è l'area in cui trasferire i byte letti
- `n` è il numero di caratteri da leggere
- in caso di successo, restituisce un intero positivo ($\leq n$) che rappresenta il numero di caratteri effettivamente letti

Scrittura di File: system call `write`

```
int write(int fd, char *buf, int n);
```

- `fd` è il file descriptor del file
 - `buf` è l'area da cui trasferire i byte scritti
 - `n` è il numero di caratteri da scrivere
-
- in caso di *successo*, restituisce un intero positivo che rappresenta il numero di caratteri effettivamente scritti

8.5 File System di Windows

- Il file system delle ultime versioni di Windows (*NTFS - Native NT File System*) è di tipo transazionale con capacità di recupero dopo una caduta del sistema (*crash*).
- Fornisce il supporto al concetto di partizioni logica (*volume*).
- Ogni *volume* è suddiviso in *cluster* (insieme di settori fisici del disco) che rappresentano le unità elementari di allocazione dei dati.

8.5 File System

- I file sono organizzati in directory secondo la classica struttura ad albero.
- È possibile creare collegamenti (sia hard che simbolici) ai file dando luogo a strutture più complesse quali grafi aciclici.
- Ogni file è caratterizzato da un insieme di *attributi*

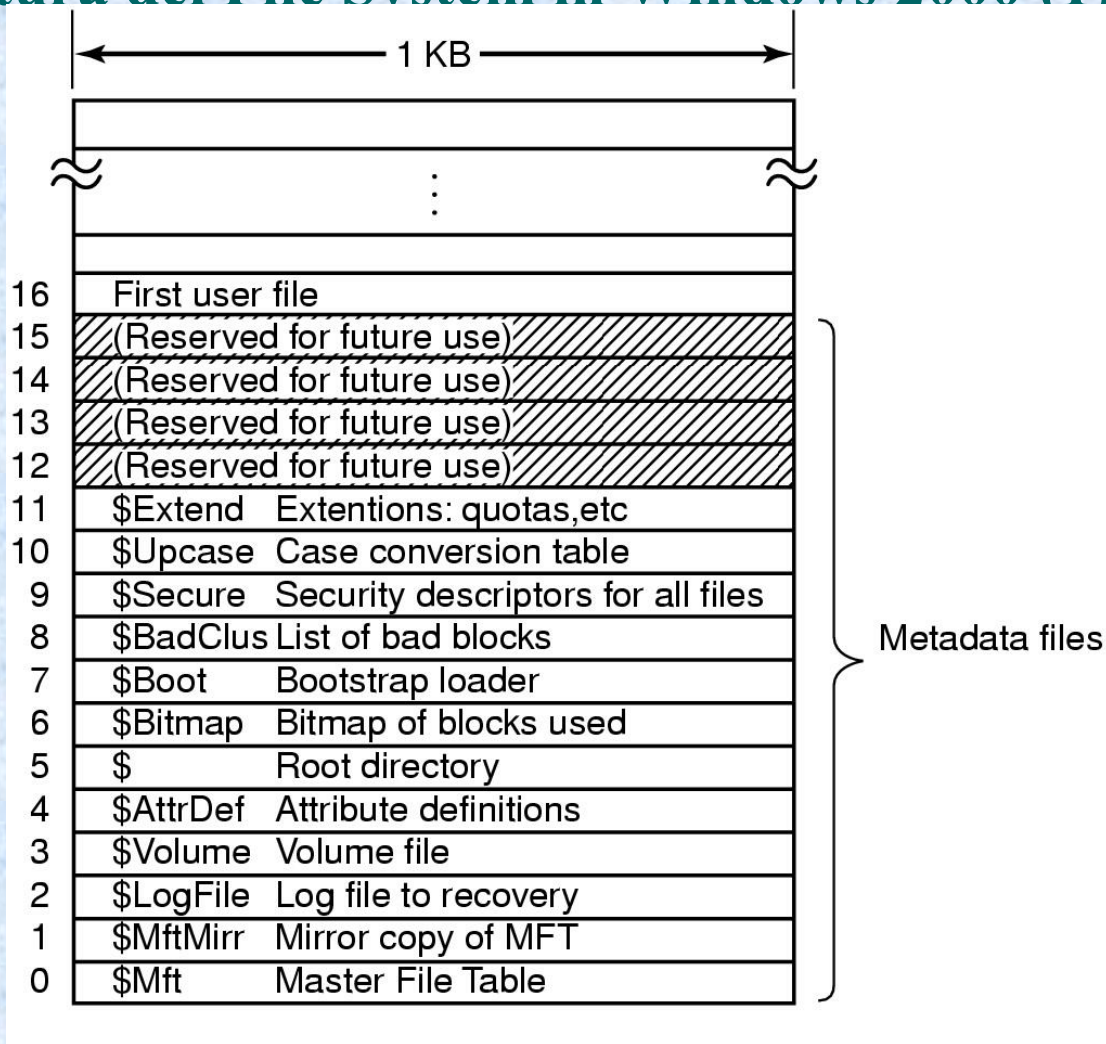
8.5 File System

- Attributi di un file:
 - ✓ nome;
 - ✓ percorso assoluto (*absolute path*);
 - ✓ dimensioni;
 - ✓ diritti di accesso;
 - ✓ vari tipi di statistiche;
 - ✓ ‘logged stream’
 - ✓ contenuto del file;
 - ✓

8.5 File System

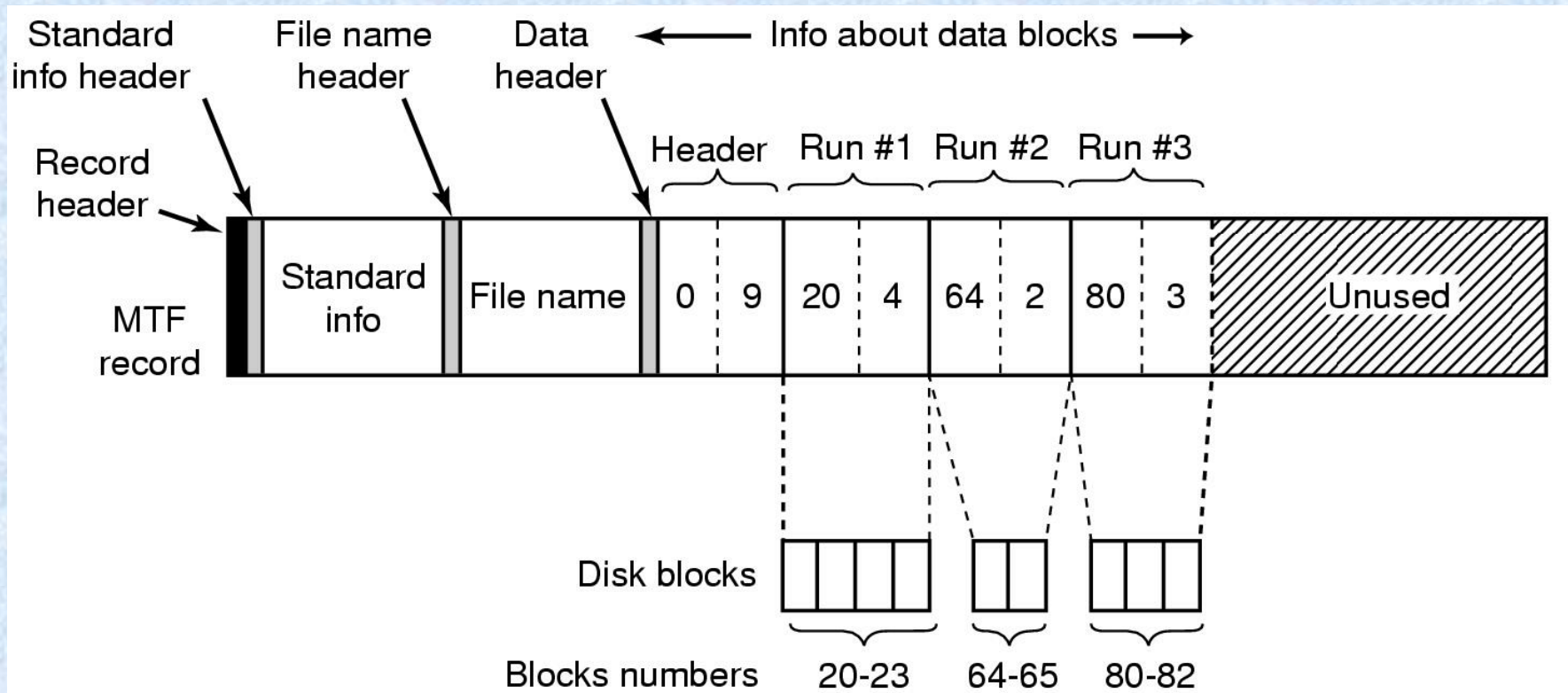
- Ogni file viene descritto da uno o più *record* contenuti in un file speciale detto *MTF* (*Master File Table*).
- Gli attributi di piccole dimensioni sono contenuti direttamente nella *MTF* (*attributi residenti*).
- Gli attributi di grandi dimensioni, come i contenuti del file, sono memorizzati in una serie di cluster sul disco (*estensioni contigue*).

Struttura del File System in Windows 2000 (1)



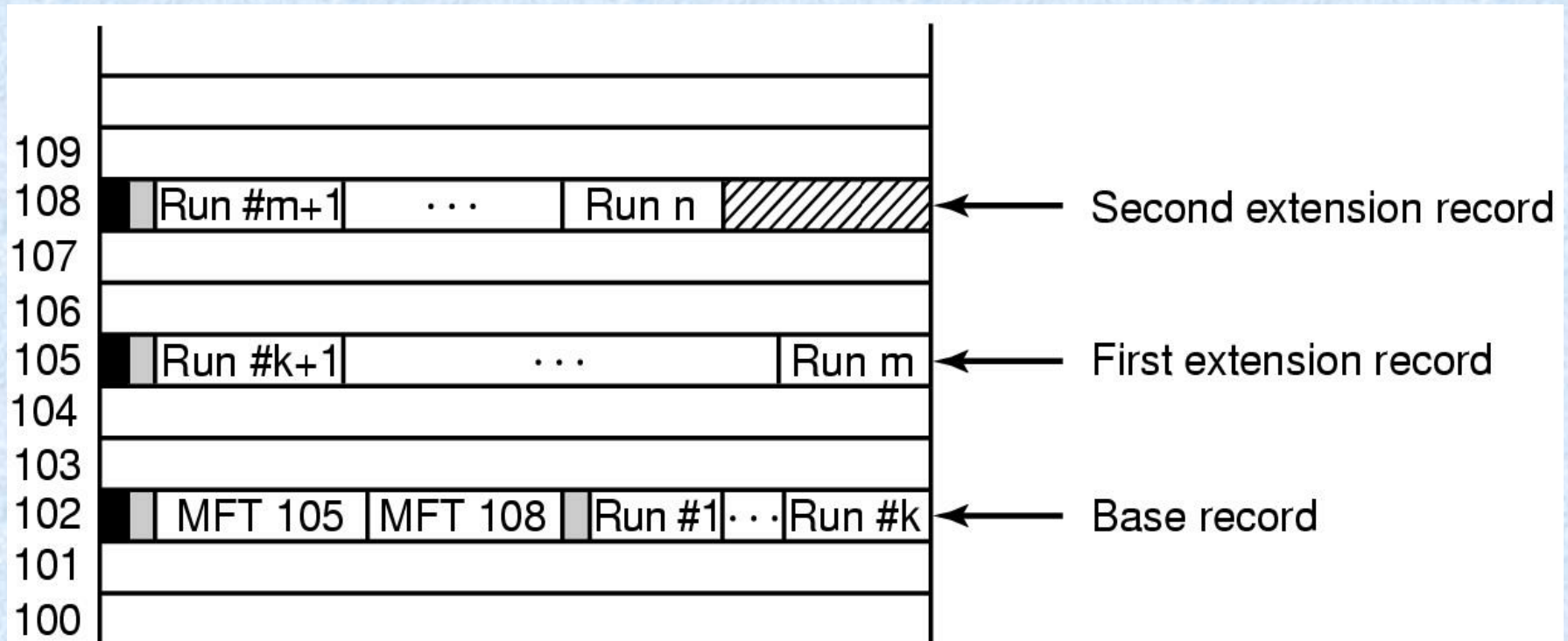
La master file table di NTFS

Struttura del File System in Windows 2000 (3)



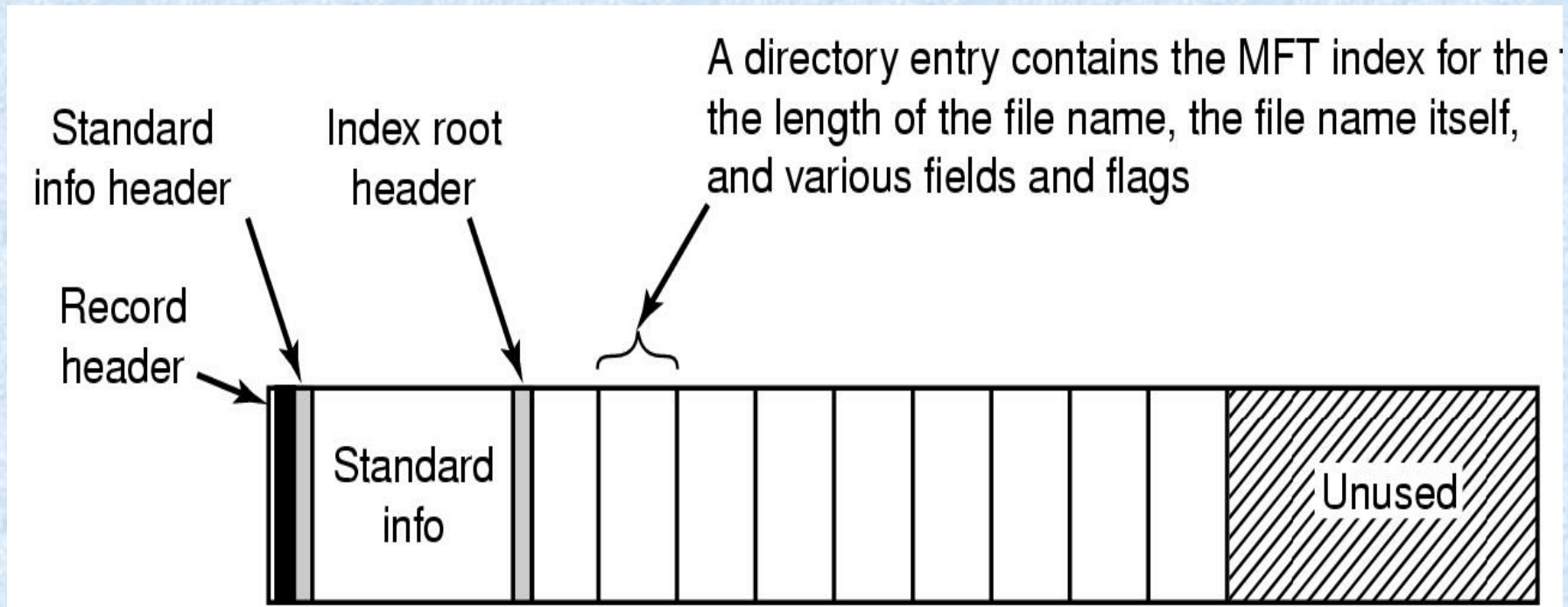
Un record MFT per un file di 3 *run* e 9 blocchi

Struttura del File System in Windows 2000 (4)



Un file che richiede 3 record MFT per memorizzare i suoi *run*

Struttura del File System in Windows 2000 (5)



Il record MFT di una piccola directory.